

A REWARD BOOK



\$10.95



introduction to

**TRS-80
LEVEL II BASIC**

**and
computer
programming**

michael p. zabinski

**introduction to
TRS-80 LEVEL II BASIC
and computer programming**

introduction to TRS-80 LEVEL II BASIC and computer programming

MICHAEL P. ZABINSKI, Ph. D.

Professor

Fairfield University

PRENTICE-HALL, INC., Englewood Cliffs, New Jersey 07632

Library of Congress Cataloging in Publication Data

Zabinski, Michael P

Introduction to TRS-80 LEVEL II BASIC and computer programming.

Includes index.

1. TRS-80 (Computer)—Programming. 2. BASIC (Computer program language) I. Title.

QA76.8.T18Z32 001.64'2 80-15015

ISBN 0-13-499962-2 (pbk.)

ISBN 0-13-499970-3 (case)

© 1980 by PRENTICE-HALL, Inc.,
Englewood Cliffs, New Jersey 07632

*All rights reserved. No part of this book
may be reproduced in any form or
by any means without permission in writing
from the publisher.*

Printed in the United States of America

10 9 8 7 6 5 4 3

Editorial/production supervision
and interior design by Linda Mihatov
Cover design by RL Communications
Manufacturing buyer: Joyce Levatino

TRS-80 is the registered trademark of Radio Shack, a Division of Tandy Corporation.

PRENTICE-HALL INTERNATIONAL, INC., *London*
PRENTICE-HALL OF AUSTRALIA PTY. LIMITED, *Sydney*
PRENTICE-HALL OF CANADA, LTD., *Toronto*
PRENTICE-HALL OF INDIA PRIVATE LIMITED, *New Delhi*
PRENTICE-HALL OF JAPAN, INC., *Tokyo*
PRENTICE-HALL OF SOUTHEAST ASIA PTE. LTD., *Singapore*
WHITEHALL BOOKS LIMITED, *Wellington, New Zealand*

*To
Toby,
Eric,
and
Marion*

contents

Preface	xi
List of Examples	xiii
Introduction	1
1	
Your TRS-80 Computer	3
1.1 Getting Started	3
1.2 The Keyboard	3
1.3 Communicating with the Computer	4
2	
Specifying Information	6
2.1 Numerical Information	6
2.2 Character Information	7
2.3 Variables	8
2.4 Assignment	9
Exercises 1	12
2.5 Arithmetic Functions	13
2.6 Hierarchy of Arithmetic Operations	14
2.7 Variable Types	16
Exercises 2	19
3	
Computer Programs	23
3.1 Writing a Program	23
3.2 Program Clarity, Displaying Messages and Comments	28
Exercises 3	29
3.3 Keyboard Response: Input	31
3.4 Editing Computer Programs	33
3.5 Debugging Programs	37
3.6 Stop and Continue Execution	39
3.7 Saving a Program on Cassette Tape	40
3.8 Declaring Variable Types	41
Exercises 4	42

4

Decisions

46

- 4.1 Relational and Logical Operations 46
- 4.2 Flowcharts 49
 - Exercises 5 52
- 4.3 Transfer Statements 53
- 4.4 On Error Go To 59
 - Exercises 6 52

5

Looping

66

- 5.1 Loop Structure 66
- 5.2 IF-THEN Loops 66
- 5.3 FOR-NEXT Loops 70
 - Exercises 7 75
- 5.4 Subscripted Variables 79
- 5.5 Nested Loops 82
- 5.6 Multiple Subscripts 84
- 5.7 Debugging Loops: Tracing and Playing Computer 85
 - Exercises 8 89

6

Input-Output

93

- 6.1 READ and DATA Statements 93
- 6.2 Formatting Output 97
- 6.3 Cassette Input-Output 107
 - Exercises 9 109

7

Library Functions

112

- 7.1 INT Function 112
- 7.2 RND Function 114
- 7.3 More Functions 116
 - Exercises 10 119

8

Subroutines

122

- 8.1 The Purpose of Subroutines 122
- 8.2 Unconditional Transfer to Subroutines 122
- 8.3 Conditional Transfer to Subroutines 125
- 8.4 A Final Comment 128
 - Exercises 11 128

9

Graphics 131

- 9.1 Background 131
- 9.2 Graphing with SET (X, Y) 131
- 9.3 Other Graphics Functions 137
- Exercises 12 139

10

Strings 142

- 10.1 Review 142
- 10.2 ASCII Codes and Related Functions (ASC and CHR\$) 143
- 10.3 Character Manipulation Functions (LEN, LEFT\$, RIGHT\$, MID\$) 146
- 10.4 Other String Functions 148
- Exercises 13 155

APPENDICES

I

Error Messages 159

II

Reserved Words 161

III

BASIC Glossary 162**Solutions to Even-Numbered Exercises** 171**Index** 181

preface

The computer language, BASIC, short for Beginner's All-purpose Symbolic Instruction Code, was developed at Dartmouth College by John Kemeny and Thomas Kurtz in 1964. Since then it has gained acceptance in industry, research, and education. Its use is continuously growing as computer manufacturers, and especially home computer manufacturers, recognize its simplicity and popularity. The Radio Shack TRS-80 microcomputer communicates in BASIC. This book provides an introduction to TRS-80 Level II BASIC and, through it, to some of the important concepts and applications of computer programming.

The reasons for the increased use of BASIC are many. Persons interested in learning practical computer use find BASIC to be a fast way to reach their goal. Because BASIC has a conversational, interactive nature and a simple structure, it is an attractive teaching tool with which computer programming concepts can be presented to beginners.

Just as the English language has many dialects, so does BASIC have many different implementations. The TRS-80 has two versions of BASIC commonly referred to as Levels I and II. Level II is more extensive than Level I and is the subject matter of this book. Level II BASIC contains a powerful arithmetical facility along with a large library of common mathematical and nonmathematical functions. It has many error diagnostics, which produce easily understandable error messages and excellent editing features. Level II also includes comprehensive graphic capabilities and numeric accuracy of up to 16 digits. These attributes combined with good input-output procedures make Level II BASIC an ideal computer language.

This book is directed at the beginning programmer. With a small set of instructions, the beginner can very quickly begin to write elementary computer programs. We take advantage of this feature of BASIC and emphasize those parts of the language that are most frequently used. Many examples are included to illustrate the use of BASIC and to demonstrate how a computer can be programmed to perform many different tasks. In addition, an abundance of exercises is presented at the end of each chapter with solutions to selected problems given at the end of the book. The reader is encouraged to try as many exercises as possible, for computer programming is best learned by doing.

The level of presentation and selection of material make this book suitable for a wide variety of readers. No previous experience with computers and no mathematical background beyond the basic arithmetic skills are assumed. The material is presented as it would appear on the TRS-80 video display. A unique feature of this book is the inclusion of explanatory com-

ments alongside the computer displays. These comments highlight and reinforce the text discussion by pointing out new procedures and emphasizing important points. The reader would therefore benefit by working along with the text, trying out the illustrative examples on the computer, and experimenting with variations. The way to learn a new computer language is to use it. Each student should run as many programs as he has time for so as to acquire a feel for this new tool.

This is a textbook suitable for the self-learner or for a one-semester introductory course. It should not be viewed as a reference manual in BASIC. The primary purpose of the book is to introduce Level II BASIC as well as computer-programming concepts. It is for the nonspecialist who wants to learn to use the TRS-80 effectively.

I am greatly indebted to many people who have contributed to the development of this book. Particular thanks are due to Stephen Cline, editor, Prentice-Hall, and to David Gunzel, director technical publications, Radio Shack. I also express my appreciation to Ann Hannon of Greenwich High School and to the teachers of the Orange Public Schools for reviewing the manuscript and providing helpful suggestions. In addition, I wish to thank John Donovan, Richard Golden, Edward Hanlon, Howard Hecht, and John McCann for their support in the preparation of this text and my wife whose advice and typing were magnificent. Programming for the chapter exercises in this book was done—brilliantly—by Michael Galaty.

MICHAEL P. ZABINSKI, Ph. D.

list of examples

Chapter 2

Interchange Values of Variables 11
Compound Interest 14
Metric Conversion 16
Single Versus Double Precision 18
A Divisibility Check 18
Rounding to the Nearest Penny 19

Chapter 3

Walking Kitty Corner 27
An Arithmetic Trick 32
Mortgage Payments 38
Temperature Conversion 40

Chapter 4

Find the Largest Among Three Numbers 56
Producing Variable Displays 58
Data Validity Checks 60
Change for a Dollar Bill 61

Chapter 5

The Rule of 72 68
Evaluating an Infinite Series 71
Averaging a Set of Numbers 73
How Fast Can the Computer Add? 74
The Legend of the Wise Old Man 75
Find the Largest Element of an Array 80
The Multiplication Table 82
Magic Squares 84
Sorting a List of Numbers in Descending Order 86

Chapter 6

Unit Pricing 95
Practice Your State Capitals 95
A Sales Report 98
Graphing an Equation 100
Pascal's Triangle 101
Checkbook Balancing 105

Chapter 7

Long Division 112
Rounding to Any Desired Accuracy 113
Generating Random Numbers Between Given Limits 114
Tossing Heads and Tails 115
Random Graphic Display 115
A Bar Graph of the ABS Function 116
The Rule of 72 Verified 118

Chapter 8

Producing a Blinking Display 124
Computer-Assisted Instruction 125

Chapter 9

Graph of Degrees Fahrenheit and Celsius 133
Bar Graph of a Frequency Count 135
Random Walk 137

Chapter 10

Maintaining a Status Message on the Screen 145
Number of Words in a Text 147
Palindromes 147
Coding a Message 148
Underlining a Title 150
Binary-to-Decimal Conversion 150
Character String Entry Routine 152
Shoot the M's, A Video Game 153
Shoot the Duck 153
Etch-a-Sketch 154

introduction

The computer revolution is here! Today, computers are invading virtually every aspect of life in America. They are changing the way we work, play, and even think. That they have affected the lives of nearly all of us is indisputable. Their impact has been experienced in areas as widely separated as space research and primary instruction. We live in a world that is increasingly dependent on computers. These machines not only calculate our paychecks and our bank balances, but they are invading areas where their application once seemed inconceivable. The physician may use the computer to diagnose a case, the attorney to research a legal matter, the policeman to investigate the records of an alleged criminal, and a history teacher to simulate the Civil War. It now seems improbable that any career will remain untouched by computers, as computers are penetrating directly or indirectly into nearly every aspect of human affairs.

The seemingly unlimited usefulness of the computer itself has been augmented by its rapid development. The number of computers produced has risen dramatically. In the early 1950s there were less than 1000 computers in existence. By 1976 the number of computers had risen to 200,000. This phenomenal growth is primarily due to the major scientific advances in computer design. The first electronic computer, the ENIAC, which was huge and filled an entire room, was built in 1946. It consisted of 18,000 vacuum tubes, and tended to overheat and break down. Since that time, with the advent of transistors, integrated circuits, and the silicon chip, computers have become smaller, more reliable, faster, and less expensive. We distinguish between computers, minicomputers, and microcomputers. Around 1965 the least expensive computers were called minicomputers. In general, they are less powerful, cheaper, and smaller, although there is considerable overlap. To some extent history repeated itself with the advent of the microcomputer. In the early 1970s, very low cost computer products began to appear and were called microcomputers. The prefix micro- applies to the very small size as compared with a mini. Once again there is considerable overlap between micro- and minicomputers. Minicomputers lowered the price of computers to less than \$100,000 and brought the computer into the laboratory and the manufacturing plant's production line. Microcomputers reduced the price of computers to below \$1,000 and brought the computer into the small business and the home.

Regardless of size and price, all computers possess common characteristics of design and performance. Typically, a computer consists of five units: input, output, memory, arithmetic, and control. Information must be fed into the computer as **input**, for example, via the keyboard or punched cards.

The data or instructions are stored in the computer's **memory**. Computer storage can be visualized as a set of post-office boxes with each box capable of holding a single number or character of information. These boxes are arranged in such a way that their contents can be easily reached or accessed. Memory holds the information received, the commands to be followed, and the results of work accomplished. The actual calculations take place in the **arithmetic** unit. Besides calculating, a computer can make comparisons to determine whether two quantities differ. Such comparisons are also made in the arithmetic unit. The results of the computer's work are displayed through the **output** unit. In the case of the TRS-80, the display may be on the screen or on the printer. The **control** unit coordinates the flow of data. Like the conductor of an orchestra, it coordinates the activities of the units of the computer to ensure proper processing.

A basic characteristic of the computer is that it has no inherent intelligence. It does not understand any human language; nevertheless, a computer can act upon instructions given in a language that is well suited to tasks the computer can accomplish. The language of the TRS-80 is BASIC; it allows you to create instructions using familiar English terms. Another important characteristic of a computer is its ability to carry out instructions extremely quickly. This fact, along with the observation that many tasks involve repeating similar operations, has led to the concept of a **computer program**. A program consists of a set of instructions provided to the computer in advance of actual computations. By first giving a computer all the instructions needed to accomplish a task, and then telling it to start executing the instructions, the computer can work at its own tremendous speed, repeating particular operations as many times as necessary to get the job done.

The language of BASIC, the Beginner's All-purpose Symbolic Instruction Code, was born in 1964. It was developed at Dartmouth College by Kemeny and Kurtz. The BASIC language is oriented to conversational use at the computer. The idea was to make the language syntax very easy to learn and use. In 1967, Kemeny and Kurtz reported that they had introduced some 2,000 students at Dartmouth to BASIC, indeed a very basic computer language.

chapter 1 | your TRS-80 computer

1.1 GETTING STARTED

The TRS-80 computer consists of four units. The **power supply**, the **keyboard**, the **video monitor**, and the **cassette recorder**. The keyboard is used to type information into the computer. The information we type in and the computer's responses are displayed on the screen. The computer itself is inside the keyboard. The cassette recorder is used to load from tape programs into the computer or to record programs on tape. These units need to be connected and plugged in carefully. Follow the detailed instructions that accompany the computer, and be sure you get all the plugs to fit properly. To turn on the computer, press the power button on the video display and the power button on the back of the keyboard. Once the power is turned on, the red light on the keyboard lights up and the display

```
MEMORY SIZE?
```

appears on the screen. Press the white key labeled ENTER. The computer responds

```
RADIO SHACK LEVEL II BASIC  
READY  
>-
```

The computer is now ready for your instructions in Level II BASIC. Computers equipped with Level I BASIC display the READY immediately after power has been turned on.

1.2 THE KEYBOARD

BASIC is a conversational computer language that enables us to carry on a dialog with the computer. We talk to the computer by using a typewriter-like keyboard. We type our messages and transmit them to the computer. At the same time, the information we type also appears on the screen.

The keyboard is divided into two zones. The first zone is the primary part of the keyboard; it contains all the keys necessary to operate the computer. The second zone, located on the right side of the keyboard, contains a duplicate set of numeric keys and a second white ENTER key. It is a

calculator-style numeric keypad that makes typing of numbers convenient and efficient for those who are accustomed to calculator usage. Some earlier model keyboards do not have a numeric keypad. The letters of the alphabet shown on the keys print only in capitals. The SHIFT key need not (but may) be pressed for the letters to be displayed in capitals. The letter O and the number zero should not be confused since they are not interchangeable. To avoid confusion, the number zero is slashed (Ø).

Some of the keys are shared by two characters. To type the upper character press the SHIFT key. This is just like typing capitals on a regular typewriter. For convenience the keyboard has two SHIFT keys located at the right and left ends of the bottom row of keys. Aside from the usual characters, such as letters, digits, addition, subtraction, and punctuation, several characters are peculiar to BASIC. The multiplication sign is a star (*), the division sign is a slash (/), and the exponent is an up-arrow (↑). In addition, there are other special-purpose keys, for example, #, @, \$, and %. These will be introduced later. Finally, several keys located along the right and left edges of the keyboard are used to manipulate where on the screen the information is typed. These include the CLEAR, →, ←, BREAK, ↑ and ↓ keys.

1.3 COMMUNICATING WITH THE COMPUTER

When we first turned on the computer we pressed ENTER to obtain the READY followed by ➤—. The — is called the **cursor**. We now do some typing. Type I AM HAPPY. The space bar is used for spaces between words. Notice how the cursor moves to the right. The cursor indicates where the next character will appear on the screen. Suppose we typed by mistake the word HPPY (the A is missing). To erase the text, we press the ← key. Each time we press the key one character is erased. The shift ← (press the SHIFT and ← keys at the same time) erases the entire line at once. Erase it.

Each line on the screen can hold up to 64 characters. Enter any text and count them. As you type briskly, you may press a second key before you have released the first. At first type slowly to be sure no extraneous characters are typed. Now erase the line and type in the entire ABC in double letters, AABBCDDDEE . . . Press shift → and notice that the letters are suddenly twice the size and that every other letter of the AABBC . . . text has been deleted. Each letter of the ABC now only appears once. Continue typing digits and other characters. Up to 32 characters fit on a line. When the line is filled, the cursor automatically moves to the next line. To erase the line, press shift ←. The enlarged cursor remains. Additional typing will continue to appear in the 32 character per line format. To return to the normal type size, press CLEAR. This key clears the screen and places the smaller cursor at the top left corner of the screen. We can now type 64 characters per line.

Press the ↓ key. It moves the cursor down along the left edge. When it is at its lowest position, type your name. Then press the ↓ key 15 times and watch your name move up to the top of the screen. There are 16 lines on the screen. Repeat this exercise. Press CLEAR and then press ↓ to move the cursor to the bottom left corner of the screen. Now press 1, ↓, 2, ↓, 3, ↓, . . . 15, ↓, 16. Subsequently, continue to type in ↓, 17, ↓, 18, and notice how the first few numbers disappear from the display. Press CLEAR to erase the screen and to place the cursor at the top left corner.

So far we have not truly communicated with the computer. We have typed information onto the screen but did not transmit it to the computer. To transmit information to the computer, the ENTER key must be pressed. Type your name and press ENTER.

	COMMENTS
TRS-80	Type in a name and press ENTER.
?SYNTAX ERROR	Computer responds: Syntax error.
READY	
>_	The cursor is waiting.

Oops! What happened? The computer did not understand our entry and therefore responded with the error message. This is one of many types of error messages that the computer uses to inform you of errors. To avoid such error messages, we must adhere to the rules of BASIC. We must learn how to properly communicate with the computer.

chapter 2 | specifying information

The great power of computers lies in their ability to handle a large amount of information rapidly. We begin our study of BASIC by looking at some of the types of information that we can use. The two types of information discussed in this chapter are **numerical information** and **character information**. We can request that the computer display information on the screen by using the instruction **PRINT**.

2.1 NUMERICAL INFORMATION

Numbers such as 18, 157, or 89 can represent information of importance to you, such as your age, weight, and grade on your last exam. Decimal numbers are expressed in the usual manner, for example, 18.5 or 3.14159.

A number such as 18.5 is a positive number. It could therefore be written equally well as +18.5. The plus sign is optional. In the case of a negative number, we must place the minus sign before the number.

COMMENTS

```
PRINT 3. 0
3
PRINT 3. 120
3. 12
```

You type in PRINT followed by a number. Then press ENTER. That number is then displayed by the computer on the screen. Note that trailing zeros after the decimal point are dropped, and for values less than one, the zero before the decimal point is also deleted. The + sign is not printed; the - sign is printed. A number without a sign is always positive.

```
PRINT +3. 12
3. 12
```

Positive numbers are displayed with a leading blank instead of the plus sign.

```
PRINT -3. 12
-3. 12
```

Negative numbers are displayed without a leading blank.

```
PRINT -0. 3120
-. 312
? 3
3
```

Leading and trailing zeros are not displayed.

The ? is an **abbreviation** for PRINT. In contrast to Level I BASIC, Level II allows only for very few abbreviations. For clarity, we do not use the abbreviated form.

REMEMBER: Always press ENTER to transmit your message to the computer.

BASIC uses a modified version of **scientific notation** for representing very large numbers or very small numbers. Scientific notation breaks a number into two parts, a number between 1 and 10 and an exponent of 10. For example, the speed of light is about 300 million meters per second, which can be written as 300,000,000 or in scientific notation as 3×10^8 . In BASIC we can express this numerical information as the number 3E8, where E indicates that 8 is the exponent of 10. The following examples illustrate scientific notation.

Number	Scientific Notation	BASIC
1230000	1.23×10^6	1.23E+06 ^a
0.000123	1.23×10^{-4}	1.23E-04

^aBASIC replaces “times ten to the” by E.

COMMENTS

PRINT 100000 100000	Once the ENTER key is pressed the computer displays 100000.
PRINT 1000000 1E+06	At 1 million, scientific notation is used.
PRINT 1E+5 100000	
PRINT 1E6 1E+06	1E6 is a valid numeric constant.
PRINT 0.01 .01	
PRINT 0.001 1E-03	Scientific notation is used by the computer for numeric constants less than 0.01.

2.2 CHARACTER INFORMATION

In addition to numerical information, BASIC also allows for the use of character information in the form of **character strings**. Character strings may consist of a single character such as

“A”, “B”, “C”, “2”, “3”, “*”, “+”, “=”

or several characters such as

“ABC”, “23”, “*+=”

Character information must be enclosed in quotation marks.

COMMENTS

<code>PRINT "A"</code>	Character data are typed in with
<code>A</code>	quotes. On output, the computer
<code>PRINT "23"</code>	displays character data without
<code>23</code>	the quotes.

REMEMBER: Character information is always enclosed in quotes. It may contain up to 255 characters.

The following examples illustrate additional features of character information.

<code>PRINT " "</code>	<i>COMMENTS</i>
	Request that a blank be printed.
<code>PRINT</code>	The computer displays a blank;
	double spacing.
	Print nothing.
<code>PRINT "HE SAID "TO BE OR NOT TO BE""</code>	The computer again skips a line;
<code>HE SAID</code>	displays a blank.
<code>?SN ERROR</code>	Try to display a quote.
	Cannot have more than one set of
	quotation marks.
<code>PRINT "HE SAID 'TO BE OR NOT TO BE'"</code>	Rest beyond first set of quotes is
<code>HE SAID 'TO BE OR NOT TO BE'</code>	ignored.
	So we use apostrophes within the
	quotes.
<code>PRINT "THIS WORKS</code>	The closing quotes are missing.
<code>THIS WORKS</code>	They are not required.

The syntax error (SN ERROR) encountered above is one of many different error messages that the computer uses to denote an illegal procedure. A complete list of the error messages is given in Appendix 1.

2.3 VARIABLES

The previous sections introduced different types of information; thus far it has been necessary to enter information from the keyboard. We often want to store information in the computer for later use. This capability is provided through variables. A **variable** is a name that represents data stored in the computer. It may be numerical or character data. It is not fixed, but may be changed whenever desired.

We want to compute the interest that a savings account will earn in one year. We need to have the account balance at the start of the year. This value is not fixed; it may vary from person to person. We assign this quantity a name, for example **B**. The quantity **B** is called a **variable**. For one person, **B** may be 1000; for another it may only be 100. So **B** varies but always represents the balance at the start of the year. Another variable that we need in order to compute the interest paid by the bank is the interest rate. We can name it **R**.

Notice that we used the variables **B** and **R** to represent the account Balance and the interest Rate. It is a good idea to select variable names that are closely associated with the quantity they represent. For example, use the variable name **M** for Mary's age and **J** for John's age.

Variable names are formed from alphabetic characters and numerals. Variable names must start with a letter (A–Z) and may be followed by letters or digits (0–9). Examples of valid **numerical variable** names are

A A3 AB Z5

Examples of invalid numerical variable names are

A+ 3A

The **A+** contains an illegal character; the **3A** does not begin with a letter.

String variables contain character information; numerical variables contain numerical information. String variable names are formed by appending a dollar sign (\$). The maximum number of characters that may be included in a character string is 255 characters. Examples of valid **string variable** names are

A\$ A9\$ DC\$

Examples of invalid string variable names are

A+\$	The + is an illegal character.
8C\$	The variable name cannot begin with a character other than a letter.
XY	The \$ is missing; this is a legal numerical variable name.

In Level I BASIC only two string variables are available. These are variables **A\$** and **B\$**.

REMEMBER: The name of a string variable always ends with a \$.

2.4 ASSIGNMENT

The procedure by which we specify the value of a variable is called **assignment**. Assignment is indicated by the equal sign (=). To assign the value 1000 to the variable **D**, we use the instruction **D=1000** or **LET D=1000**. The **LET** is optional.

Type in **D**, the equal sign, and **1000**. Then press ENTER. From this point on, **D** has the value 1000. The value of **D** can subsequently be changed through another assignment statement. To display the current value of the variable **D**, we type in **PRINT D** and press ENTER.

	<i>COMMENT</i>
LET D=1000	Assign D the value 1000.
PRINT D	Request display of the value of D.
1000	The present value of D is 1000.

REMEMBER: The LET is optional; we will omit it.

When the computer encounters the assignment statement `D=1000`, it places the value 1000 in a storage location in its memory and associates this storage location with the variable name D. The assignment instruction in itself does not cause the value of the variable D to be displayed. A separate `PRINT` is needed. Then printing the value of D does not change its value in storage.

	<i>COMMENTS</i>
LET D=2000	The variable D is now 2000. Note the assignment does not cause the value of the variable to be displayed.
PRINT D	
2000	The value of D is displayed.
PRINT D	
2000	The value of D is still 2000.

Spaces (blanks) within statements such as between the `PRINT` and variable D are optional. Spaces are generally included only for readability.

Variables are initially set to zero by the computer. So unless otherwise specified, all variables have zero values to start with. In Level I BASIC, the value of an undefined variable is unpredictable. For example, we have not as yet assigned the variable R a value. So we expect the value of R to be zero. The LET is optional, so we omit it.

	<i>COMMENTS</i>
PRINT R	R has not been assigned a value and is therefore zero; the space between PRINT and R is optional.
0	
R=.07	The assignment statement does not cause a display by the computer.
PRINT R	
.07	R is now .07.
R=0	R is assigned the value zero.
PRINT R	
0	R is again zero.

Variable names may exceed two characters, but only the first two characters are used by the computer. The variables NU and NUMBER are therefore one and the same. The computer does not recognize the letters beyond the U. Also, variable names may not contain words used in the BASIC language, such as ON, FOR, or NEXT. There are many such keywords commonly referred to as **reserved words**. A list of the BASIC reserved words is given in Appendix 2.

	COMMENTS
NUMBER =1	Variable names more than two
PRINT NUMBER	characters in length are permis-
.1	sible.
PRINT NU	NU and NUMBER are the same
1	variables. Only the first two char-
	acters are recognized.
NU=10	NU is specified as 10.
PRINT NUMBER	
10	Now NUMBER is also 10.
GONE=7	GONE is an illegal variable. It con-
?SN ERROR	tains the BASIC reserved word
	ON. GO

The assignment is not always just the usual equality. For example,

	COMMENTS
PRINT A	A has not been assigned a value.
0	Like all undefined variables, it
	starts out as zero.
A=1	A is assigned the value 1.
PRINT A	
1	The old value of A (=0) has been
	replaced by 1.
A=A+2	The value of A is replaced by its old
PRINT A	value (=1) plus 2.
3	A is now 3 (= 1 + 2).

We may think of the assignment as a replacement. Even though in arithmetic $A = A + 2$ is not a valid relationship, in BASIC it is. We have in this example introduced for the first time an arithmetic operation, the addition. In the next section we discuss arithmetic operations in detail.

The assignment statement works similarly with strings.

	COMMENTS
N1\$="TOBY"	The character string TOBY is as-
PRINT N1\$	signed to the variable N1\$.
TOBY	
N2\$="MARION AND ERIC"	N2\$ is the name of a string vari-
PRINT N2\$	able whose value is MARION
MARION AND ERIC	AND ERIC.
N1\$=N2\$	Variable N1\$ is assigned the value
PRINT N1\$	of variable N2\$.
MARION AND ERIC	The value of N1\$ is no longer
PRINT N2\$	TOBY.
MARION AND ERIC	The value of N2\$ remains un-
	changed in the process.



Example: Interchange Values of Variables

Two variables A and B have been assigned values (for example, $A=10$ and $B=20$). Write a sequence of instructions to interchange the values of A and B without simply stating $B=10$ and $A=20$.

	<i>COMMENTS</i>
A=10	A and B are given values.
B=20	
PRINT A	
10	A is 10.
PRINT B	
20	B is 20.
A=B	Set A equal to B.
PRINT A	
20	A is now 20 as desired.
B=A	Set B equal to A.
PRINT B	B is 20, but we want it to be 10;
20	this is because B = A sets B equal
	to the current value of A (=20).

So we introduce a third variable, C:

	<i>COMMENTS</i>
A=10	A is 10 to start with.
B=20	B is 20 to start with.
C=A	Set C equal to A.
A=B	Set A equal to B.
B=C	Set B equal to C, which equals the
PRINT A	original value of A, that is, 10.
20	A is now 20 as desired.
PRINT B	
10	B is now 10 as desired.

REMEMBER: When a variable is assigned a new value, its old value is lost.

EXERCISES 1

In the following exercises, record the result you anticipate in the column Anticipated Display. If the actual display differs from the anticipated display, record the result in the column Display with an explanation. Refer to Appendix 1 for error messages.

1. Fill in the Anticipated Display, enter the instructions, record the results, and explain.

Instruction	Anticipated Display	Display
a. PRINT 3. 1		
b. PRINT 12345678901		
c. PRINT -0. 00000123		
d. PRINT 15E2		
e. PRINT -35E-2		
f. D=10		
PRINT D		
g. PRINT "D"		
h. PRINT E		
i. PRINT "D D D"		
j. PRINT "ABCD		
k. ?D		

2. Use the PRINT statement to display your age.
3. Use the PRINT statement to display your name.
4. Use two PRINT statements to display your name and home address.
5. Enter the following instructions. If you anticipate an error, indicate the source of that error.

Instruction	Anticipated Display	Display
a. PRINT MY NAME		
b. PRINT "HE SAID, "I AM HAPPY""		
c. A\$="TERRIFIC"		
d. ACE=1		
e. A=5		
f. A+3=9		

6. Specify and then display:
 - a. A string variable that contains the letters of the alphabet without spaces between the letters.
 - b. A string variable containing all the digits from 0 to 9.
7. a. Find the largest N for which 10^N does not exceed the limit for a constant on your computer. (*HINT*: In scientific notation, enter successively increasing numbers of the form 1E2, 1E10, and so on.)
 - b. How does the computer indicate when a number is too big to be used? (This is called **overflow**.)
 - c. Try entering successively smaller numbers, such as 1E-2, 1E-10, and so on. What happens?

2.5 ARITHMETIC FUNCTIONS

Addition (+), subtraction (-), multiplication (*), division (/), and exponentiation (\uparrow) are the five available arithmetic functions. (Level I BASIC does not have exponentiation.) On the screen the symbol for exponentiation appears properly as an arrow pointing up (\uparrow). The printer, however, prints this symbol as a square bracket ([).)

	COMMENTS
PRINT 1+2	The instruction is entered.
3	The sum is displayed.
PRINT 1-2	Subtracting 2 from 1 yields -1.
-1	
A=5	
PRINT 9*A	The constant 9 is multiplied by the variable A. The result is 45.
45	
B=10	
PRINT B/A	One variable is divided by the other.
2	10 divided by 5 is 2.
PRINT 5[2	Five squared is 25. On the printer, exponentiation shows up as a bracket ([).
25	
PRINT 4[0.5	The square root of 4 is 2; the exponent 0.5 is equivalent to the square root.
2	

The instruction `PRINT 1+2` differs from `PRINT "1"+"2"`:

```
PRINT "1"+"2"
12
```

Character strings are not added; they are **concatenated** (strung together).

```
PRINT "1"+"2"+"3"
123
```



Example: Compound Interest

Suppose you deposit \$1000 in a savings account. What will be the interest paid and the balance of your account after 1 year and after 10 years? Assume the money is compounded once a year and earns an interest of 7%.

	<i>COMMENTS</i>
<code>I=1000*.07</code>	Compute the interest paid the first year.
<code>PRINT I</code>	The interest is \$70.
<code>70</code>	Balance at the end of the first year is the interest earned added onto the initial deposit.
<code>B=1000+I</code>	The balance at the end of the first year is \$1070.
<code>PRINT B</code>	Combining the above steps yields the balance in one computation.
<code>1070</code>	Compute the balance after 10 years.
<code>B=1000*1.07</code>	The bracket indicates exponentiation.
<code>PRINT B</code>	After 10 years the balance is \$1967.15.
<code>1967.15</code>	The difference between the final balance and the initial deposit of \$1000 is the total interest paid over 10 years.
<code>PRINT B-1000</code>	
<code>967.15</code>	

In the above example we introduced the idea of more than one arithmetic function appearing in a single instruction. In the computation `B=1000*1.07[10]` we have a multiplication (*) and an exponentiation ([]). The order in which these arithmetic operations is performed is most important. In this case the exponentiation is first, followed by the multiplication.

REMEMBER: On the screen the symbol for exponentiation is \uparrow ; on the printer and in this book it is `[]`.

2.6 HIERARCHY OF ARITHMETIC OPERATIONS

Arithmetic operations are generally evaluated from **left to right**, keeping in mind these rules:

1. Whatever is inside the innermost set of **parentheses** is always evaluated first. The next innermost parentheses are evaluated next, and so on.

2. The arithmetic functions have the following three levels of precedence: exponentiations (raising to a power) before all others, then multiplications and divisions, and finally additions and subtractions, which are performed last.

3. Within each level the operations are performed from left to right.

The following examples demonstrate these rules (assume $X = 1$, $Y = 2$, and $Z = 3$):

Expression	Value	Comments
$X + Y - Z$	0	+ and - have equal hierarchy, so compute from left to right.
$X + Y * Z$	7	Multiplication first followed by addition.
$X * Z / Y$	1.5	* and / have equal hierarchy, so compute from left to right.
$Z * (X + Y)$	9	Parentheses are evaluated first, followed by the multiplication.
$Z / Y + Y / X$	3.5	First Z/Y , then Y/X , then the addition.
$Z * (X / (Y + Z))$.6	The innermost parentheses are evaluated first: $Y + Z = 5$. Next the outer parentheses are evaluated. $X / (Y + Z) = 1 / (2 + 3) = 1/5 = 0.2$. Finally the multiplication is carried out: $3 * 0.2 = 0.6$.

REMEMBER: In case of a tie in precedence, the sequence of operations is left to right.

It is important to note that the presence of parentheses may or may not affect the value of the expression. If the parentheses do not alter the value of the expression, the parentheses are said to be **redundant** and are included for clarity alone. For example,

Expression	Value	Comments
$Z / Y + Y / X$	3.5	As explained above.
$(Z / Y) + (Y / X)$	3.5	The parentheses are redundant; they only help clarify the sequence.
$Z / (Y + Y) / X$.75	Now the parentheses affect the value. First the () are evaluated ($Y + Y = 4$); then from left to right, $Z/4/X = 3/4/1 = 0.75/1 = 0.75$.

Another rule to remember is that arithmetic operations can follow one

another without being separated by parentheses:

	COMMENTS
PRINT 5-(-2) 7	The parentheses can separate the two arithmetic operations.
PRINT 5--2 7	The same result without the ().



Example: Metric Conversion

Write a sequence of instructions to convert the height 5 feet 10 inches to inches and to centimeters.

	COMMENTS
F=5	5 feet.
I=10	10 inches.
IN=F*12+I	Compute the inches: first the multiplication, and then the addition is performed.
PRINT IN 70	5 feet 10 inches equals 70 inches.
CM=(F*12+I)*2.54	The () are evaluated first; then multiply by 2.54 to convert inches to centimeters.
PRINT CM 177.8	

REMEMBER: (1) Every expression must contain the same number of right and left parentheses. (2) The product AB must be written as A*B.

2.7 VARIABLE TYPES

We have distinguished between variables that store numerical values and variables that store character information. Numerical values can be stored in **integer**, **single-precision** and **double-precision** modes. These three different types of variables store numbers with different degrees of precision. Single-precision variables are accurate to 6 **significant figures**; double-precision variables are accurate to 16 significant figures. Integer variables store only whole numbers in the range -32768 to +32767. Each type of variable has a corresponding character that specifies the type. String variables are declared by means of a \$, for example, AB\$. The maximum number of characters that can be stored in a string variable is 255. Single- and double-precision variables are declared by means of a ! and a #, respectively, for example, AB! and AB#. Integer variables are declared with the % symbol, for example, AB%. The variable types are summarized in Table 2.1. The limits on single-precision numeric constants are -1.701411E+38 to +1.701411E+38; double-precision numbers must be in the range -1.701411834544556D+38 to +1.701411834544556D+38. The D represents the exponent in double-precision scientific notation.

TABLE 2.1 Declared Variables

Variable Type	Identification Character	Limits on Magnitude	Examples
Single Precision	!	-1.701411E+38 to +1.701411E+38 inclusive	AA! = 1.23456E+38; A! = -1.23456; six digits displayed
Double Precision	#	-1.701411834544556D+38 to +1.701411834544556D+38	B# = 1.234567890123456D+38; 16 digits displayed
Integer	%	-32768 to +32767 inclusive	II% = 32767; ACE% = -32000
String	\$	Up to 255 characters	TEXT\$ = "TO BE OR NOT TO BE"

REMEMBER: When a variable is not declared as an integer, a string, or single or double precision, it is automatically given single precision.

The following examples illustrate the variable types:

```
AB=1. 234564
PRINT AB
1. 23456
AB=1. 234567
PRINT AB
1. 23457
```

```
AB=1234567
PRINT AB
1. 23457E+06
```

```
AB=0. 001234567
PRINT AB
1. 23457E-03
```

```
AB!=1. 234567
PRINT AB!
1. 23457
```

```
AB!=1234567
PRINT AB!
1. 23457E+06
```

```
AB#=1. 2345678901234567
PRINT AB#
1. 234567890123457
AB#=12345678901234567
PRINT AB#
1. 234567890123457D+16
```

```
AB%=1234567
PRINT AB%
?OV ERROR
```

COMMENTS

AB is a seven-digit single-precision number.

Six digits are displayed. The sixth digit is rounded off.

The sixth digit is rounded up.

AB is a seven-digit number.

Six digits are displayed, rounded, and in scientific notation.

AB is a ten-digit number.

Leading zeros are not significant.

Declare AB! as a single-precision variable.

Rounded and six significant figures are displayed.

Declare AB! as a single-precision variable. E indicates single-precision scientific notation; six significant figures are displayed in scientific notation.

Declare AB# as a double-precision variable.

The sixteenth digit is rounded.

A 17-digit number. Sixteen significant figures are displayed. The D indicates double-precision scientific notation.

Declare AB% as an integer variable. Overflow error occurs since AB% exceeds 32767.

	<i>COMMENTS</i>
AB\$="12345678901234567"	Declare AB\$ as a string variable.
PRINT AB\$	AB\$ contains 17 characters.
12345678901234567	All 17 characters are displayed.

REMEMBER: The variables AB, AB\$, AB!, AB#, and AB% are all different variables.



Example: Single Versus Double Precision

Multiply in both single and double precision 14593 by 846 and 2220247 by 8.

	<i>COMMENTS</i>
A#=14593*846	A# is a double-precision variable.
PRINT A#	The multiplication is in double
12345678	precision with all eight digits displayed.
A=14593*846	Single-precision multiplication; an-
PRINT A	swer rounded to six significant
1.23457E+07	figures.
A=2220247*8	Single precision.
PRINT A	Due to roundoff, only five digits
1.7762E+07	are displayed.
A#=2220247*8	Double precision.
PRINT A#	An American Bicentennial date.
17761976	



Example: A Divisibility Check

Integer variables only retain the integer portion of a number and ignore the decimal fraction. **Integer arithmetic** therefore differs from conventional arithmetic in that the result is always an integer.

	<i>COMMENTS</i>
A=7	
B=2	
AB=A/B	
PRINT AB	
3.5	
AB%=A/B	AB% is an integer variable.
PRINT AB%	Only the integer portion of the
3	fraction A/B is retained.

Variable B divides into variable A evenly if AB equals AB%. In the above example AB does not equal AB%, so 7 is not divisible by 2.

COMMENTS

```

A=6
B=2
AB=A/B
AB%=A/B
PRINT AB
3
PRINT AB%
3

```

AB equals AB%. So A is divisible by B; 6 is divisible by 2.

*Example: Rounding to the Nearest Penny*

COMMENTS

```

D=1.236
C=100*D+0.5
PRINT C
124.1

CX=C
PRINT CX/100
1.24

```

D equals \$1.236.
 Converts dollars to cents and adds 0.5 cents.
 $100 \times 1.236 + 0.5 = 123.6 + 0.5 = 124.1$.
 Drop fractional cents; truncate C.
 Convert to dollars and cents; $124/100 = 1.24$.
 \$1.236 is rounded to \$1.24.

In this example we take advantage of integer arithmetic. We must remember that C% cannot exceed 32767. In the event that it does, a different approach must be taken in rounding. It involves the integer INT function, which is discussed in a later chapter. Does the above method properly round down? Try $D = 1.234$ for which the expected result is 1.23.

EXERCISES 2

1. Evaluate each of the following expressions; be sure you understand how each result is obtained. Refer to Appendix 1 for explanations of error messages.

Instruction	Anticipated Display	Display
a. PRINT 3+2-5	_____	_____
b. PRINT 3+2*5	_____	_____
c. PRINT 3+2/5	_____	_____
d. PRINT (3+2)*5	_____	_____
e. PRINT 3*5-2	_____	_____
f. PRINT 1+3*5-2	_____	_____
g. PRINT 2*(9-2)/(10-3)	_____	_____
h. PRINT 2*(9-2)/((10-5)*2)	_____	_____
i. A=3, B=5, C=10, A\$="A", B\$="B", C\$="C", D=2E-10, A#=3, _ D#=2E-10	_____	_____
PRINT A/B	_____	_____
PRINT (C-B)*A	_____	_____
PRINT (B+1)*C/B	_____	_____
PRINT A-B*C	_____	_____
PRINT C/A	_____	_____
PRINT -C/A	_____	_____
PRINT (-C)/A	_____	_____
PRINT 4/A/2	_____	_____
PRINT A\$+B\$+C\$	_____	_____
A%=A/B	_____	_____

Instruction	Anticipated Display	Display
PRINT A%	_____	_____
PRINT D	_____	_____
PRINT D#	_____	_____
PRINT A+D	_____	_____
PRINT A#+D#	_____	_____
?A	_____	_____
A%=32767	_____	_____
PRINT A%+1	_____	_____
A%=32768	_____	_____
PRINT A%	_____	_____
A%=0.01	_____	_____
PRINT A%	_____	_____
B%=-0.1	_____	_____
PRINT B%	_____	_____
j. PRINT 1E38	_____	_____
PRINT E38	_____	_____
PRINT 1/1E38	_____	_____
PRINT 1/3!	_____	_____
PRINT 1/3#	_____	_____
PRINT 1/3!+1/3#	_____	_____
PRINT 2%+1/3#	_____	_____

- Multiply in single and double precision: 4869684.5 times 8.
- Is `LET A + B = C` a valid BASIC statement? Why or why not? Once you have answered the question, enter the statement into the computer.
- Write a minimal (as few characters as possible) BASIC expression for each of the following:

Conventional Notation	BASIC Notation
a. $A + B - C$	_____
b. $C \times (A - Y)$	_____
c. $(A \times B) - 2$	_____
d. $A \times (B \times C) \times A$	_____
e. $C - \frac{B}{3} - D$	_____
f. $\frac{A \times B - C \times (A - B)}{3} - (A \times C)$	_____

- Convert the following from BASIC notation to conventional notation:

a. $X/Y-Z$	_____
b. $Z*Y-Z*X$	_____
c. $Z/X+Y+X$	_____
d. $X*Y-X*Y$	_____
e. $X*Y*(Z+X)/X-Y$	_____

- Let $AB = 5$ and $AC = 6$. Print the sum, the difference, the product, and the quotient of AB and AC .
- Print the square, the cube, the square root and the cube root of 2. (*Note:* the cube root corresponds to an exponent of $1/3$ or 0.333333.)
- What is the displayed value? (LET $A = 5$.)

a. PRINT A/0	_____
b. PRINT A*0	_____
c. PRINT A\0	_____
d. PRINT 0*0	_____
e. PRINT 0/0	_____

9. The grades on your last five tests were 80, 90, 95, 63, and 75. Find their average.
10. Mr. Jones joined the programmer's union and now earns \$9.50 per hour for a 40-hour week and overtime at a rate of $1\frac{1}{2}$ times his normal rate (\$14.25) for any hours worked beyond 40 hours. Compute the weekly total gross pay and the overtime gross pay (if any) for the following hours worked.
 - a. 40 hours
 - b. 37 hours
 - c. 47 hours

11. Hero's formula for the area of a triangle is

$$\text{area} = \sqrt{S \times (S - A) \times (S - B) \times (S - C)}$$

where A, B, and C are the lengths of the three sides of the triangle and S is one-half of the perimeter; give BASIC expressions to evaluate Hero's formula, assuming

- a. The lengths of the sides are 3, 4, and 5.
 - b. The lengths of the sides are stored in the variables S1, S2, and S3.
12. Given a time duration in T hours and M minutes, write a BASIC expression to express this in seconds. Print your answer for 1 hour and 30 minutes, and then also for 0 hours and 1 minute.
 13. The following expression converts degrees Celsius (C) to degrees Fahrenheit (F):

$$F = \frac{9}{5}C + 32$$

Print the degrees Fahrenheit equivalent to -10, 0, 10, and 20 degrees Celsius.

14. Compute the interest on \$24 compounded annually at 3% since 1627, the year Manhattan was bought for \$24 from the Indians. The bank balance at the end of N years is $P \times (1 + I)^N$, where I represents the annual interest rate (as a decimal; for example, 7% is 0.07) and P the initial bank deposit.
15.
 - a. How long does it take to double your money in the bank? Assume annual interest rates of 5%, 7%, and 9% and an initial balance of \$1000.
 - b. Does the time to double your money change when the initial deposit is varied from \$1000 to \$5000?
16. The amount of the monthly payment necessary to pay off a home mortgage is given (in conventional notation) by

$$P = A \times I \times \frac{(1 + I)^N}{(1 + I)^N - 1}$$

where A is the amount of the mortgage and I is the monthly interest rate (as a decimal). For example, if the annual interest rate is 7%, the monthly interest rate in decimal form is 0.07/12. N is the number of monthly payments. Give a BASIC expression that is equivalent to this formula. Assuming A has a value of \$30,000, evaluate and print this expression for

- a. A lifetime of 20 years (480 months) at (yearly) interest rates of 8% and 10%.
 - b. An interest rate of 10% and a lifetime of the mortgage of 20 and 30 years.
17. As N gets larger, what value does the following expression take on:

$$\left(1 + \frac{1}{N}\right)^N = ?$$

Let $N = 10, 100, \text{ and } 1000$.

18. The positions of points on a graph are given in terms of their X (horizontal) and Y (vertical) coordinates. Suppose the coordinates of the first point are specified as X1, Y1, and the coordinates of the second are X2, Y2.
- a. Give a BASIC expression to print the slope of a line passing between these two points. The slope of the line is given as the ratio of the difference in their Y coordinates to the difference of their X coordinates:

$$\text{slope} = \frac{Y2 - Y1}{X2 - X1}$$

- b. Give an expression to print the distance between the two points. The distance is given by

$$[(X1 - X2)^2 + (Y1 - Y2)^2]^{1/2}$$

Use these expressions to compute the slope and distance for the following three sets of points:

(1, 2) and (4, 8)

(3, 4) and (5, 4)

(-1, 2) and (-1, 6)

chapter 3 | computer programs

In the last chapter the compound interest example demonstrated the **PRINT** instruction along with the arithmetic operations. We computed the annual interest and the end-of-year balance of our savings account. We also determined the balance after 10 years. This set of instructions is quite useful, and we may wish to use it several times. Therefore, it may be desirable to save this set of instructions rather than to reenter it every time. This can be done by writing a **computer program**. The computer program will make it possible to determine the balance in the savings account for different interest rates, different initial deposits, and different durations, without having to retype all the instructions for every computation. To do this, we combine the instructions into a computer program.

3.1 WRITING A PROGRAM

To illustrate how to write a program, we will copy from the last chapter the instructions to compute the interest earned and the end-of-year balance for a given initial deposit and a specific interest rate. We store the Deposit in variable **D** and the interest Rate (in decimal form) in **R**.

	<i>COMMENTS</i>
D=1000	Initial deposit specified.
R=0.07	Interest rate specified in decimal form.
I=D*R	Compute the interest earned.
PRINT 1	Print the interest earned.
70	The interest is 70.
PRINT D+I	Compute and print the end-of-year balance.
1070	

In performing these calculations, the computer is in the **command mode**. Whenever the **>—** appears on the screen, the computer is in the command mode.

We want to show how to write the above instructions in the form of a computer program so that they can be saved and do not have to be reentered every time we wish to make these computations. To enter a program, we type in a **line number** followed by the instruction.

	<i>COMMENTS</i>
30 I=D*R	Line number 30 calculates the interest.
40 PRINT I	Display the interest.
50 PRINT D+I	Calculate and display the balance.
60 END	Last line of the program.

This program consists of four **statements**. Each statement has a line number; each statement may be up to 255 characters in length. The **END** statement is the last statement. We notice that, even though we are in the command mode, no results are displayed as each statement is entered. We distinguish in the command mode between the **immediate mode** and the **programming mode**. In the immediate mode, statements have no line numbers; in the programming mode they do.

To run the program, we assign a value to the variables **D** and **R**. We may give them any values we want. We then execute the program by typing **RUN** and then pressing the **ENTER** key. The **RUN** command places us in the **execution mode**. Computer programs are executed in the execution mode.

	<i>COMMENTS</i>
D=1000	D and R are specified in the immediate mode.
R=.07	
RUN	Execution of program requested.
0	Oops, the interest is zero.
0	The balance is also zero.

What happened to our very first program? The **RUN** command caused execution of the program stored in memory. But at the same time the **RUN** also executes a **CLEAR**, a command that resets all the numeric variables to zero and all string variables to null. So in our case the variables **D** and **R** are reset to zero by the **RUN** command. To avoid this complication, we rewrite the program by adding to it the following two lines:

```
10 D=1000
20 R=.07
```

This is a form of **editing**, which will be discussed in more detail later. We now execute the program.

	<i>COMMENTS</i>
RUN	Request execution of the program.
70	The interest is displayed.
1070	The total balance is displayed.

The results are as expected. After the execution of the program has been completed, the **READY** reappears. We are again in the command mode.

	<i>COMMENTS</i>
PRINT D; I	Prints the value of D and I in the immediate mode.
1000 70	Their old values are still in memory.
10 D=10000	Enter a new line 10 in the programming mode.

RUN	Request execution; switch from
700	command mode to execution
10700	mode. Interest is now 700. Bal-
	ance is now 10700.

The semicolon in the PRINT allows us to display the values of several variables on the same line. All variables that are to be printed in this manner must be separated by semicolons. The values of D, R, I, and B are all in memory at this point and can be displayed in the immediate mode.

	<i>COMMENTS</i>
PRINT D;R;I;D+I	The semicolons must separate the
10000 .07 700 10700	variables.

- REMEMBER:** (1) Whenever the **>-** appears, we are in the command mode.
 (2) In the command mode we distinguish between the programming and immediate modes.
 (3) In the immediate mode, statements have no line numbers.
 (4) In the programming mode, statements must have line numbers; line numbers allowed are 0 to 65529 inclusive.
 (5) The RUN statement switches from the command mode to the execution mode.

Line numbers must be integers (whole numbers) between 0 and 65529. Each line in the program must have a number, and the execution of the program proceeds in ascending order of line numbers. The line numbers of successive instructions can be spaced. As a matter of fact, a spacing of 10 is recommended so that additional instructions can be sandwiched in at a later time. What is the spacing between line numbers in the savings account program we wrote? To take another look at the program, we type in LIST and ENTER. A listing of the program appears on the screen.

	<i>COMMENTS</i>
LIST	Request a listing of the program.
10 D=10000	Successive line numbers differ by
20 R=.07	10 in this particular program.
30 I=D*R	
40 PRINT I	
50 PRINT D+I	
60 END	The END is optional and may be
	omitted.

The LIST command instructs the computer to display the lines of the program that are in memory. The various versions of the LIST command are

<i>COMMAND</i>	<i>COMMENTS</i>
LIST	Entire program is displayed.
LIST 20	Line 20 of the program is displayed.
LIST 20-	All lines as of line 20 are displayed.
LIST -20	All lines up to and including line 20 are displayed.

	COMMENTS
LIST 20-80	All lines 20 through 80 inclusive are displayed.
LIST.	The line just entered or edited is displayed.

Since the **END** statement is optional, let us delete the **END** in line 60 of the above program. To delete line 60, we type in 60 and press ENTER.

	COMMENTS
60 and press ENTER	Delete line 60 of the program.
LIST 60	Request a display of line 60.
READY	Line 60 no longer exists, so just the
>-	READY and the cursor appear.
60 END	The END statement is reinserted.
DELETE 60	Another way to delete a line: type
	DELETE and the line number.
LIST 60	Line 60 has again been deleted.
READY	
>-	
RUN	Execute the above program without
700	the END statement.
10700	The answers are the same as before.

The **DELETE** command erases program lines from memory. It is possible to delete individual lines or a sequence of lines.

COMMAND	COMMENTS
DELETE 60	Delete line 60.
DELETE -60	Delete all program lines up to and including line 60.
DELETE 40-60	Delete all lines starting with line 40 and including line 60.
DELETE.	Delete the line just entered.

A convenient command to use during programming is the **AUTO** command. During program entry, this command **AUTO**matically displays the next line number. All you have to do is enter the actual BASIC statement. This line-numbering function is invoked by typing in **AUTO**. The numbering begins with line 10 and continues thereafter in increments of 10. Lines 20 and 30 then follow. The various versions of the **AUTO** command are

COMMAND	SEQUENCE OF LINE NUMBERS
AUTO	10, 20, 30, ...
AUTO 50	50, 60, 70, ...
AUTO 50, 5	50, 55, 60, ...

The starting line number and the increment can be specified. To stop the automatic line-numbering function, press the **BREAK** key. The **READY** sign appears. At this point, additional line numbers may be entered or the program may be run. In the event that **AUTO** displays a line number already in use, an asterisk appears alongside the number. Press the **BREAK** key if you do not wish to reprogram that line.

To **erase** the program from memory, we type in **NEW**. This command clears the screen, displays **READY**, and deletes the entire current program from memory. In the process it resets all numeric variables to zero and string variables to null. Then if we request execution by entering **RUN**, no results are displayed:

	<i>COMMENTS</i>
RUN	Request execution.
700	The interest earned.
10700	The end-of-year balance.
NEW	Erase the program in memory.
RUN	Execution requested.
	No results; the program is gone.
PRINT D; R; I; D+I	The values of the variables have
0 0 0 0	been set to zero by the RUN command.

REMEMBER: Before typing in a program, enter **NEW** to delete the program already in memory.

The variable **MEM** contains the number of unused bytes of memory. This variable can be used in the immediate and programming modes.

	<i>COMMENTS</i>
NEW	Erase memory.
PRINT MEM	
15572	The number of available bytes is 15572.
A=1	Specify A.
PRINT MEM	Now 15565 bytes remain; it took 7
15565	bytes to specify A = 1.

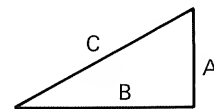
The total number of available bytes depends on the computer's memory. A 16K computer has 15572 bytes available; a 4K computer has 3284 bytes.



Example: Walking Kitty Corner

According to the Pythagorean theorem, the length **C** of the hypotenuse of a right triangle is given by the expression

$$C = \sqrt{A^2 + B^2}$$



where **A** and **B** are the lengths of the other two sides. Write a program to compute and **PRINT** the hypotenuse **C** given the sides **A** and **B**. Also compute and print the difference between the sum of the sides **A + B** and the hypotenuse **C**. This difference represents the distance saved in walking kitty corner across an intersection. A program to perform these calculations is shown below. Remember to type in **NEW** before entering the program.

	<i>COMMENTS</i>
12 A=3	Specify A as 3.
14 B=4	Specify B as 4.
16 C=(A ² +B ²) ^{0.5}	C is computed from A and B; taking the square root corresponds to raising to the exponent 0.5.
18 PRINT C	Line numbers are picked freely but in ascending order.
20 PRINT A+B-C	
22 END	
 RUN	Execution is requested.
5	The hypotenuse is 5 and the difference (A + B) - C is 2; (4 + 3 - 5 = 2).
2	

In this program we first specify A and B and then compute C. Once C is computed, the results are printed in lines 18 and 20. In order to make another computation with different numbers, we must change the values of A and B. New values are assigned to A and B by changing lines 12 and 14 in the program.

	<i>COMMENTS</i>
12 A=12	Change lines 12 and 14.
14 B=5	
RUN	Now execute with a different A and B.
13	C = 13.
4	A + B - C = 12 + 5 - 13 = 4.

REMEMBER: In memory the statements of a program are stored in line-number order. The sequence in which the lines are typed is not important.

3.2 PROGRAM CLARITY, DISPLAYING MESSAGES AND COMMENTS

It is important that a program be written in a way that is easily understood by the reader. A good program is one that proceeds smoothly and logically from beginning to end. A good program therefore requires good planning.

Another factor influencing the readability of a program is the complexity of the BASIC expressions that are used. Because of the large number of built-in functions that are available and the ability to chain expressions together (we will see that later), some amazing calculations can be accomplished in one line of code. The argument that such expressions execute more rapidly is not really relevant if the milliseconds of saved computer time are gained at the expense of minutes or hours of the programmer's and reader's time.

Program clarity is also enhanced by the appropriate choice of variable names and the use of **explanatory comments** within the program. Comments are indicated by **REM**. They are nonexecutable statements that merely improve the readability of the program through explanatory comments.

Another aspect of clarity deals with the output of the results. It is often useful to display messages along with the numerical results of a calculation. It helps us to recall what the results mean. We can display messages by dis-

playing a character string as a step in our program or on the same line by placing a semicolon between the character string and the variable. For example, in the kitty corner program we now enter into the computer

```
17 PRINT "THE HYPOTENUSE IS"
20 PRINT "DIFFERENCE IN LENGTHS IS"; A+B-C
```

Since the rest of the program is still in the computer, line 17 is added to it, and the new line 20 replaces the old version of line 20. We also add a comment REMinding us that variables A and B are the legs of the right triangle.

```
10 REM A&B ARE THE LEGS OF A RT. TRIANGLE
```

Like all other statements, a REM statement may be up to 255 characters in length. The **abbreviation** for REM is the apostrophe.

```
10 ' A&B ARE THE LEGS OF A RT. TRIANGLE
```

This version of line 10 is identical to the above. For clarity we do not use the abbreviated form of REM.

We now list the program and execute it.

	COMMENTS
LIST	
10 ' A&B ARE THE LEGS OF A RT. TRIANGLE	Request a listing of the program. The reminder comment.
12 A=12	
14 B=5	
16 C=(A ² +B ²) ^{0.5}	The bracket means exponentiation. Line 17 was properly inserted.
17 PRINT "THE HYPOTENUSE IS"	
18 PRINT C	
20 PRINT "DIFFERENCE IN LENGTHS IS"; A+B-C	The new version of line 20 is now part of the program.
22 END	
RUN	Execution is requested.
THE HYPOTENUSE IS	
13	
DIFFERENCE IN LENGTHS IS 4	The difference 12 + 5 - 13 = 4.

The character strings and numerical values are printed in one case on separate lines and in the other case on the same line. Note the blank space between the word IS and the number 4. For a negative result the minus sign takes up this space.

REMEMBER: REM statements make a program easier to follow. Character information with PRINT statements makes the output results self-explanatory.

EXERCISES 3

1. Give the display you expect from executing the following examples. Check your results by entering them in the immediate mode. Watch out for the positions of the blanks in the expressions and their displays.

Instruction	Anticipated Display	Display
a. PRINT 5	_____	_____
b. PRINT 5+6	_____	_____
c. PRINT 5; 6	_____	_____
d. PRINT "FIVE"; 6	_____	_____
e. PRINT "FIVE"; "SIX"	_____	_____
f. PRINT "FIVE"; -6	_____	_____
g. PRINT "FIVE "; X	_____	_____
h. PRINT "1 2 3 "; "4"	_____	_____
i. X=12	_____	_____
PRINT "A DOZEN IS "; X	_____	_____
j. PRINT "TWO DOZEN IS" 2*X	_____	_____
k. Y=8	_____	_____
l. PRINT X; Y	_____	_____
m. PRINT "X="; X; "Y="; Y	_____	_____
n. PRINT MEM	_____	_____
o. NEW	_____	_____
p. PRINT MEM	_____	_____
10 PRINT 5 / THIS IS A TEST	_____	_____
RUN	_____	_____

2. Write the proper BASIC expressions to produce the following output. Use the variables A = 1, B = 10, and C = 100 to produce numeric output. (For example, the 111 in part c is obtained by displaying A + B + C.)

- MY COMPUTER LIKES ME
- 100 YEARS AGO
- 111 YEARS AGO
- 1 PLUS 10 EQUALS 11
- HE HAS -10 DOLLARS

3. The variables M, D, and Y are used to store today's date. Write the statements that will express the date as --/--/-. The date June 11, 1984, will then appear as 6/11/84. Repeat the process with variables A\$, B\$, and Y\$.
4. Specify an opening bank balance. Compute and display the interest for one day, one week, one month, three months, half a year, and one year. Assume an annual interest rate of 8%. Change the interest rate to 12% and repeat the computations.
5. What output will the following programs produce?

- ```
20 PRINT "A BREAK TODAY"
10 PRINT "YOU DESERVE"
30 PRINT "AT McDONALD'S"a
```
- ```
10 A$="BASEBALL "
20 B$="HOT DOG "
30 C$="APPLE PIE"
40 PRINT A$+B$+C$
50 PRINT "AND CHEVROLET"b
```
- ```
10 A=6
20 LET B=1
30 PRINT A; " OF"; B; " IS"
40 ? B/2; " A DOZEN OF THE OTHER"
```

*Note:* Did you type in NEW before entering each program?

6. Write programs that will produce the following displays:

- ```
IN FRENCH RED IS ROUGE
IN GERMAN IT IS ROT
```
- ```
THE SUM OF X AND Y IS: 5
THE PRODUCT OF X AND Y IS: 6
```

*Note:* In this program let X and Y equal 2 and 3, respectively.

<sup>a</sup>"You Deserve a Break Today®" and "McDonald's®" are registered trademarks owned by McDonald's Corporation and are reprinted only by written permission.

<sup>b</sup>Reprinted by permission of General Motors Corporation.

7. Write and run a program that uses variables A, B, and C to display D if  $A = 2$ ,  $B = 3$ ,  $C = 4$ , and  $D = A*B - C$ .
8. Write a program to put in two numbers and output their quotient. What if either one of the two numbers is zero?
9. Write a program to put in three numbers and display their average.
10. Write a program to compute the volume of a cube of side S. The program should yield output in the following forms. (The volume is the side cubed. Use  $S = 10$ , and subsequently  $S = 20$ .)
  - a. THE VOLUME IS  
-----
  - b. THE VOLUME IS \*-----
  - c. THE SIDE IS\*--
  - d. THE SIDE IS \*--  
THE VOLUME IS\*-----

The \* stands for a space (blank), and the dashes are to be filled in by the values of the variables.
11. Modify the program of the previous exercise to compute and display the surface area of the cube in addition to its volume. The surface area equals six times the area of each face. The area of each face is  $S*S$ . Use  $S = 15$ . The output should be

```
THE LENGTH OF EACH SIDE IS *--
THE VOLUME IS *-----*AND THE SURFACE AREA IS*-----
```

The \* stands for a space (blank).

12. Write a program to produce the following picture of a tree.

```
 X
 XXX
XXXXX
XXXXXXXX
 XXX
 XXX
```

13. Investigate how much memory is taken up by each of the following instructions:
  - a. `A=1`
  - b. `A%=1`
  - c. `A#=1`
  - d. `A!=1`
  - e. `A$="1"`
  - f. `PRINT 1`
  - g. `10 A=1`

### 3.3 KEYBOARD RESPONSE: INPUT

In all our programs so far we emphasized the need for specifying information prior to execution. In the case of the program *kitty corner* we specified the sides of the triangle in lines 12 and 14 of the program; in the case of the compound interest program we specified the deposit and the interest rate. A more convenient way of entering this information is to do so during the actual execution. This is accomplished with the **INPUT** statement. When the **INPUT** statement is encountered during execution, the computer stops to let the user enter the necessary information via the keyboard. The **INPUT** statement can call for several numeric or string variables at once. The items

in the list must be separated by commas. We now incorporate the input statement into the compound interest program, which we presented earlier. Remember to type in NEW before entering the program.

|                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |                                                                                                                                                                                                                                                                                                                                                                                                   |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <pre> 10 REM  COMPOUND INTEREST 15 PRINT "ENTER THE INTEREST RATE, FOR EX. 0.07" 20 INPUT R 25 PRINT "ENTER INITIAL DEPOSIT &amp; NO. OF YEARS SEP.       BY A COMMA"<sup>1</sup> 30 INPUT D,N 35 B=D*(1+R)<sup>N</sup> 40 PRINT "THE BALANCE AFTER";N;"YEARS IS \$";B 45 PRINT "THE TOTAL INTEREST PAID IS \$";B-D 50 END  RUN ENTER THE INTEREST RATE FOR EX. 0.07 ? 0.07 ENTER INITIAL DEPOSIT &amp; NO. OF YEARS SEP. BY A COMMA ? 1000,10 THE BALANCE AFTER 10 YEARS IS \$ 1967.15 THE TOTAL INTEREST PAID IS \$ 967.15  RUN ENTER THE INTEREST RATE FOR EX. 0.07 ? 0.07 ENTER INITIAL DEPOSIT &amp; NO. OF YEARS SEP. BY A COMMA ? 1000,10 THE BALANCE AFTER 10 YEARS IS \$ 1967.15 THE TOTAL INTEREST PAID IS \$ 967.15 </pre> | <p><i>COMMENTS</i></p> <p>Just a REMinder.<br/>Tells you what to enter.</p> <p>Enter two items.<br/>Bracket is exponentiation.<br/>Output with appropriate messages.</p> <p>Request execution.</p> <p>The ? indicates input expected.<br/>Type 0.07 and press ENTER.<br/>You enter 1000, 1</p> <p>Request another execution.</p> <p>You enter 0.07</p> <p>The deposit is \$1000 for 10 years.</p> |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

**REMEMBER:** The INPUT statement is an excellent means for data entry during execution. It makes the program general in use.



### *Example: An Arithmetic Trick*

The computer asks you to think of a three-digit triple number (such as 222, 666, 888, and so on). You then “tell” the computer the total of the three digits (18 for 666). The computer will then reveal your original number.

```

10 CLS
20 INPUT "HI, WHAT IS YOUR NAME"; N$
30 PRINT "OK ";N$;" I HAVE AN ARITHMETIC TRICK FOR YOU"
40 PRINT "THINK OF A THREE-DIGIT TRIPLE NUMBER"
50 PRINT "TYPE IN THE SUM OF ITS DIGITS"
60 INPUT S
70 PRINT N$;" ", "YOUR NUMBER WAS ";37*S;"!"
80 END

```

<sup>1</sup>In order to keep the program and comments separate, line 25 of the above program has been split. The computer actually displays it on one line. Similarly, lines of programs are split elsewhere in the book.

```

RUN
HI, WHAT IS YOUR NAME? ERIC
OK ERIC I HAVE AN ARITHMETIC TRICK FOR YOU
THINK OF A THREE-DIGIT TRIPLE NUMBER
TYPE IN THE SUM OF ITS 3 DIGITS
? 18
ERIC, YOUR NUMBER WAS 666 !

```

The **CLS** command is introduced in line 10. CLS clears the screen. It is a good idea to start programs with the CLS command since it removes from the screen during execution all unnecessary clutter.

The above program illustrates the use of INPUT in conjunction with numeric and string variables. As input, a string may be entered with or without **quotes** around it. However, if the string contains a comma, a leading blank, or a colon, then the quotes are required.

A new form of the INPUT statement is shown in line 20. This statement is equivalent to the two statements:

```

20 PRINT "HI, WHAT IS YOUR NAME?"
25 INPUT N$

```

Upon execution of the program, the message HI, WHAT IS YOUR NAME is displayed on the screen along with a question mark. The computer adds the question mark to denote INPUT. It is therefore recommended that the message be phrased in such a way that it forms a question. Once the message appears on the screen, the user enters a value for the variable and presses ENTER. A semicolon must separate the message from the variable in the INPUT statement.

**REMEMBER:** You cannot input a string into a numeric variable, or vice versa. If you do, the computer will respond by displaying **?REDO**. It requests that all data for the particular INPUT statement be reentered.

### 3.4 EDITING COMPUTER PROGRAMS

In the process of writing computer programs we frequently make errors that need to be corrected before the program will operate properly. If the error is detected before the ENTER key has been pressed, the **←** key can be used to backspace the cursor and delete one character at a time. The **shift ←** key is used to delete an entire line. There are other ways of correcting the code if an error is detected after the ENTER key has been pressed.

Suppose the following program has been written to assign the values 5 and 10 to the variables X and Y, and then to compute and print their ratio.

|                                         | <i>COMMENTS</i>       |
|-----------------------------------------|-----------------------|
| LIST                                    | Request a listing.    |
| 10 Y=5                                  |                       |
| 20 Y=10                                 |                       |
| 30 PRINT "THE RATTIO OF X AND Y IS";X/Y | RATTIO is misspelled. |

We execute the program:

|                            | <i>COMMENTS</i>       |
|----------------------------|-----------------------|
| RUN                        |                       |
| THE RATTIO OF X AND Y IS 0 | RATTIO needs editing. |

This result is unexpected; it is incorrect. We also notice that the word RATTIO has a typographical mistake. So the program needs **editing**. We need to fix line 10 where the Y is to be replaced by X, and in line 30 the word RATTIO is misspelled. (*Note:* in Level I BASIC we would need to completely retype lines 10 and 30.)

Since line 10 is so short it makes good sense to retype the line. As for line 30, removing a letter T without retyping the entire line is helpful. In order to demonstrate editing, we now edit both lines 10 and 30.

First we switch from the command mode into the **edit mode** by typing in the **EDIT** command along with the line number to be edited. Then press ENTER.

|                         | <i>COMMENTS</i>                                                 |
|-------------------------|-----------------------------------------------------------------|
| EDIT 10 and press ENTER | Switch from the command mode into edit mode.                    |
| 10 _                    | The computer's response; the _ indicates the cursor's position. |

Now we press L (list line) without ENTER; the computer responds by listing line 10 as it is presently stored in the computer followed by 10\_.

|        | <i>COMMENTS</i>                                                |
|--------|----------------------------------------------------------------|
| 10 Y=5 | We now wish to replace the Y in line 10 by an X.               |
| 10 _   |                                                                |
| 10 X _ | Type C to indicate a character is to be changed followed by X. |

At this point we have replaced the Y by X in line 10. The remainder of line 10 is fine. Pressing the space bar moves the cursor over to the right, one space at a time, and displays the remainder of the line.

|          | <i>COMMENTS</i>                                |
|----------|------------------------------------------------|
| 10 X=5 _ | Pressing the space bar twice displays the = 5. |

Now press ENTER to record the change.

While editing, we do not press ENTER. Once ENTER is pressed the computer records the changes made in the current line and returns you from the edit mode to the command mode. Additional lines can then be edited or the program can be run.

We continue to edit our program. Next we will remove the extra T in line 30:

|                         | <i>COMMENTS</i>                     |
|-------------------------|-------------------------------------|
| EDIT 30 and press ENTER | Request edit of line 30.            |
| 30 _                    | Computer's response to the request. |

TABLE 3.1 Editing procedures

| Purpose                                                                                                                                     | Procedure                                                                                                                                                                                         |
|---------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>When in command mode</i>                                                                                                                 |                                                                                                                                                                                                   |
| Delete an entire line                                                                                                                       | Type the line number and then ENTER                                                                                                                                                               |
| Replace an entire line                                                                                                                      | Type the new version of the line and then ENTER                                                                                                                                                   |
| Switch to edit mode to edit line number <i>N</i>                                                                                            | Type EDIT <i>N</i> and ENTER                                                                                                                                                                      |
| <i>When in edit mode</i>                                                                                                                    |                                                                                                                                                                                                   |
| Switch to command mode and record changes                                                                                                   | Type ENTER                                                                                                                                                                                        |
| Save changes and switch to command mode                                                                                                     | Type E                                                                                                                                                                                            |
| Cancel changes and switch to command mode                                                                                                   | Type Q                                                                                                                                                                                            |
| Cancel changes and switch cursor to start of line                                                                                           | Type A                                                                                                                                                                                            |
| Display characters one at a time                                                                                                            | Type space bar; cursor moves to the right                                                                                                                                                         |
| Display entire line                                                                                                                         | Type L; can now edit the line                                                                                                                                                                     |
| Display rest of line                                                                                                                        | Type X; can now add to line                                                                                                                                                                       |
| Delete rest of line                                                                                                                         | Type H; can now add to line; to stop but remain in edit mode type SHIFT↑                                                                                                                          |
| Delete characters within a line                                                                                                             | Type nD to delete <i>n</i> characters starting at the current cursor position; or type I and press ← key to delete one character at a time; to escape I mode but remain in edit mode, type SHIFT↑ |
| Delete all characters from current cursor position up to <i>n</i> th occurrence of character <i>c</i> counting from present cursor position | Type nKc; if no value is specified for <i>n</i> , then all characters will be deleted up to but not including the first occurrence of character <i>c</i>                                          |
| Change <i>n</i> characters within a line                                                                                                    | Type nC to change <i>n</i> characters starting at the current cursor position                                                                                                                     |
| Insert characters within line                                                                                                               | Type I; to stop but remain in edit mode, type SHIFT↑                                                                                                                                              |
| Move cursor <i>n</i> spaces to the left                                                                                                     | Type n←                                                                                                                                                                                           |
| Move cursor to <i>n</i> th occurrence of character <i>c</i> , counting from current position                                                | Type nSc                                                                                                                                                                                          |
| Move cursor <i>n</i> spaces to the right                                                                                                    | Type n space bar                                                                                                                                                                                  |

Press L to display the entire line:

```
30 PRINT "THE RATTIO OF X AND Y IS": X/Y
30 _
```

#### COMMENTS

Computer displays line 30 and sets up for its change.

Now press the space bar to display the line one character at a time up to the first T in RATTIO.

TABLE 3.2 Editing examples

| BASIC Statement      | Correction                                             | Editing Procedure                                                                                                                                                                                                                                                                                                                                                         |
|----------------------|--------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 10 PRINT A; B; ; B*G | Change B's to G's<br>and delete the third<br>semicolon | Type EDIT 10 and ENTER.<br>Press space bar eight times<br>until the first semicolon<br>appears. Press C followed by<br>G to change the first B to a<br>G. Press the space bar once<br>to display the second semi-<br>colon. Press D to delete the<br>third semicolon. Then press<br>C and G. Press ENTER to<br>escape the edit mode and<br>return to the command<br>mode. |
| 20 PRINT L; M; -N    | Delete the minus                                       | Type EDIT 20 and ENTER.<br>Press X to display the entire<br>line. Press ← twice to erase<br>-N. Type N and press<br>ENTER.                                                                                                                                                                                                                                                |
| 30 PRINT A; B; C     | Change the B to B52                                    | Type EDIT 30 and ENTER.<br>Press 2S to search for the<br>second semicolon. Then<br>type 2152 to insert the two<br>digits. Press ENTER.                                                                                                                                                                                                                                    |
| 40 INPUT EZ, K, MZ   | Change the K to TZ                                     | Type EDIT 40 and ENTER.<br>Press SK to search for the<br>first occurrence of K. Press<br>C followed by T and I fol-<br>lowed by Z to respectively<br>change the K to T and insert<br>Z after the T. Press ENTER.                                                                                                                                                          |
| 50 PRINT LET A=5     | Delete PRINT and<br>LET                                | Type EDIT 50 and ENTER.<br>Press KA to delete all char-<br>acters up to the first occur-<br>rence of A. Press ENTER.                                                                                                                                                                                                                                                      |
| 60 INPUT A, B        | Change line number<br>60 to 65                         | Retype entire line as line 65.<br>Then delete line 60 by typ-<br>ing 60 and ENTER. Line<br>numbers cannot be changed<br>directly.                                                                                                                                                                                                                                         |

## COMMENTS

30 PRINT "THE RAT \_      Press space bar to display the line  
up to the T.

To delete the second T, press D (Delete) once and press ENTER to return to the command mode. Another way to delete the second T is to press I (Insert) followed by backspace ← once and then ENTER.

To display the current version of line 30 we enter

LIST 30  
30 PRINT "THE RATIO OF X AND Y IS"; X/Y

## COMMENTS

Request display of line 30.  
It is now correct.



We run the edited program and obtain the expected output.

```
RUN
THE RATIO OF X AND Y IS .5
```

Numerous editing features are available in Level II BASIC. (Level I BASIC's editing is very limited.) Table 3.1 summarizes editing procedures. Table 3.2 gives additional examples of editing.

**REMEMBER:** Do not hesitate to experiment with the edit procedures. Practice will make you proficient.

The editing commands make it possible to change any character within a line except the line number. To change a line number, it is necessary to delete the line and then retype the entire line with the new line number.

### 3.5 DEBUGGING PROGRAMS

Unfortunately, nobody is perfect and we do make mistakes. Naturally, an important aspect of programming and a major portion of program development time is normally devoted to isolating and correcting errors. This process is called **debugging**.

Generally, programming errors are classified as **language errors** and **logical errors**. These two types of errors differ in that language errors are detected by the computer whereas logical errors are not. Once the program has been typed in and is run for the first time, the computer will interpret each line and list the language errors in the form of diagnostic messages. Here the computer is most helpful as it displays abbreviated error codes, such as /0 ERROR for division by zero or SN ERROR (SyNtax error) along with the line number in which the error occurred. Some errors even switch execution into the edit mode. A complete list of these errors is given in Appendix 1.

Once we have corrected all the language errors, we expect to run the program and obtain the desired output. When the answers do not make sense, we must set out to locate the errors in logic. They are usually more difficult to find since the computer offers us no assistance. For example, in computing the end-of-year balance of a savings account we may inadvertently subtract the interest earned instead of adding it to the initial deposit, or we may use an incorrect formula in our financial model. Clearly, logical errors are more difficult to detect than language errors. A more systematic approach must be taken so that we can be reasonably sure that the program is free of bugs.

A useful approach is to first run the program with test data for which the results are known. If the computer's answers check out, we are ready to make a run with the actual data. Most data processing applications involve a large amount of data. Therefore, it is important that the data be checked by the program as they are entered. For example, if the data involve ages, negative ages are impossible; similarly, working a 400-hour week instead of a 40-hour week is not possible. A **data validity check** built into the program will help to avoid erroneous results due to bad data.

The process of debugging lengthy programs is facilitated by a well-

organized program that computes the variables of interest in an orderly fashion. Also, assigning variables names that suggest their meaning helps in the debugging process.



### Example: Mortgage Payments

As an example of the debugging process, we present a program that computes monthly mortgage payments. The payment is

$$P = M \times I \times \frac{(1 + I)^N}{(1 + I)^N - 1}$$

where P is the monthly payment, M is the amount of the mortgage, I is the monthly interest rate, and N is the number of monthly payments for the duration of the loan. The program we have is

```
10 REM PROGRAM COMPUTES MONTHLY MORTGAGE PAYMENTS
15 PRINT "ENTER THE AMOUNT AND YEARS OF MORTGAGE"
20 INPUT M,Y
25 PRINT "ENTER THE INTEREST RATE IN %"
30 INPUT I
35 REM CONVERT MORTGAGE DURATION FROM YEARS TO MONTHS
40 N=Y*12
45 REM CALCULATE THE MONTHLY INTEREST RATE
50 I=I/12
55 REM CALCULATE MONTHLY PAYMENTS
60 P=M*I*(1+I)^N/((1+I)^N-1)
65 PRINT "THE MONTHLY PAYMENT IS $":P
70 END
```

Having typed in the program, we will now execute it for the first time, keeping in mind that it may have language as well as logical errors. The test data we plan to use first are M = 1200, Y = 1, I = 1; a mortgage of \$1200 for one year at 1% interest rate. The expected monthly payments for 12 months (1 year) are slightly more than \$100.

```
RUN
ENTER THE AMOUNT AND YEARS OF MORTGAGE
? 1200,1
ENTER THE INTEREST RATE IN %
? 1
?SN ERROR IN 60
READY
60 _
```

#### COMMENTS

Request execution.  
Computer displays the message.  
You enter the values.  
Computer displays the message.  
You enter the value.  
During execution computer detects a syntax error and switches from the execution mode to the edit mode.

We have encountered error SN, a syntax error, in line 60, and the computer has placed us in the editing mode. We press L to display the entire line:

```
60 P=M*I*(1+I)^N/((1+I)^N-1)
60 _
```

We notice that the parentheses do not balance; we must delete the last ( in line 60. Press X to display the entire line with the cursor appearing at the end of the line.

```
60 P=M*I*(1+I)^N/((1+I)^N-1) _
```

Now press ← five times to move the cursor to the left under the last left parenthesis (, and retype from there the N-1). Press ENTER to store the line. We list line 60:

|                                                      |                                                |
|------------------------------------------------------|------------------------------------------------|
|                                                      | <i>COMMENTS</i>                                |
| LIST 60                                              | Request a display of line 60.                  |
| 60 P=M*I*(1+I) <sup>N</sup> /((1+I) <sup>N</sup> -1) | The corrected version of line 60 is in memory. |

Now that the syntax error has been corrected we again run the program.

```

RUN
ENTER THE AMOUNT AND YEARS OF MORTGAGE
? 1200,1
ENTER THE INTEREST RATE IN %
? 1
THE MONTHLY PAYMENT IS $ 161.995

```

No more language errors! We have an answer, but it is incorrect. We expected an answer slightly more than \$100. Now what? We have a logic error! Going over the program line by line we notice that the interest rate in line 50 needs to be converted to the decimal equivalent of percent; that is, we must multiply the interest rate by 0.01. We retype line 50.

50 I=I\*0.01/12 and press ENTER.

```

RUN
ENTER THE AMOUNT AND YEARS OF MORTGAGE
? 1200,1
ENTER THE INTEREST RATE IN %
? 1
THE MONTHLY PAYMENT IS $ 100.542

```

Now the answer makes sense. We have done it and can now expect the program to work for other data.

```

RUN
ENTER THE AMOUNT AND YEARS OF MORTGAGE
? 50000,25
ENTER THE INTEREST RATE IN %
? 10,5
THE MONTHLY PAYMENT IS $ 472.091

```

If during the data entry you change your mind and wish to start over, press **BREAK** to get the READY. Then type RUN and enter the desired data.

**REMEMBER:** Never take it for granted that the answers from the computer are necessarily correct.

### 3.6 STOP AND CONTINUE EXECUTION

An important debugging technique is to place several **STOP** statements within a program. The **STOP** interrupts the execution and prints the message **BREAK IN line number**. For example, if line 25 is a **STOP**, then upon execu-

tion of line 25 the message **BREAK IN 25** will appear. Once execution has been stopped it is possible to examine variables by printing their values. To resume execution after a **STOP** or a **BREAK**, type in **CONT** and press **ENTER**. If the program is modified during the **BREAK**, then execution cannot be resumed by means of the **CONT** statement. Instead, type in **RUN** and start the execution all over.



### Example: Temperature Conversion

|                                  | COMMENTS                                                                         |
|----------------------------------|----------------------------------------------------------------------------------|
| 10 INPUT "DEGREES FAHRENHEIT"; F | Input temperature in °Fahrenheit.                                                |
| 20 C=5*(F-32)/9                  | Convert to °Celsius.                                                             |
| 25 STOP                          | Stop execution here.                                                             |
| 30 PRINT F; F*.5; C; C*.5        | Print the temperatures and their square roots. Bracket is exponentiation.        |
| 40 END                           |                                                                                  |
| RUN                              | Request execution.                                                               |
| DEGREES FAHRENHEIT? 86           | Input °F.                                                                        |
| BREAK IN 25                      | Execution is stopped.                                                            |
| PRINT C                          | Examine the value of C in immediate mode. °C = 30.                               |
| 30                               |                                                                                  |
| CONT                             | Request continuation of execution.                                               |
| 86 9.27362 30 5.47723            | The temperatures and their square roots.                                         |
| DEGREES FAHRENHEIT? 0            |                                                                                  |
| BREAK IN 25                      |                                                                                  |
| PRINT C                          |                                                                                  |
| -17.7778                         | Since °C is negative, the square root cannot be taken.                           |
| RUN                              | Instead of typing in <b>CONT</b> , now type in <b>RUN</b> for another execution. |

## 3.7 SAVING A PROGRAM ON CASSETTE TAPE

Once a program executes free of any bugs, it may be desirable to save it on tape for future use. Place the cassette tape into the recorder, rewind the tape, set the digital counter to zero, and press the **RECORD** and **PLAY** buttons. The **CSAVE** command loads a program from the computer onto the tape. With this command you must specify a program name. For example

```
CSAVE "I"
```

moves the program residing in memory onto tape and calls it "I". The name of a program may be any single alphanumeric character other than the quotes. If the name is several characters in length, only the first character is recognized by the computer. So **CSAVE "INTEREST"** is equivalent to **CSAVE "I"**. Once the program has been saved on tape, read the digital counter and make a note of where on the tape the program is saved.

**REMEMBER:** To avoid confusion, write the names of the stored programs on the case of the tape.

When a program is saved on tape, there is always the possibility that the transfer is not flawless. Therefore, a subsequent loading of the program from the tape may result in errors or even possibly in a loss of the program. One possible safeguard is to save the program on tape twice, giving it two different names, for example, **CSAVE "I"** and **CSAVE "K"**.

The **CLOAD?** command offers another alternative. Once program "I" has been saved on tape using the command **CSAVE "I"**, it can be checked by the command

```
CLOAD? "I"
```

Rewind the tape and press the **PLAY** button on the recorder. The program on tape and the program in memory are compared line by line. If the programs do not compare perfectly, the message "BAD" appears. The **CSAVE "I"** command will then have to be executed again.

The **CLOAD** command will load the first program from the cassette into the computer. To ready the recorder, rewind the tape and press the **PLAY** button. However, if the desired program is not the first program on the tape but rather the fifth program whose name is "I", then the command

```
CLOAD "I"
```

will skip over the first four programs and load the program "I". This program was originally loaded on tape using the command **CSAVE "I"**. So as long as the tape is rewound, the computer can pick up the desired program and load it into memory.

### 3.8 DECLARING VARIABLE TYPES

In the previous chapter we noted that variables can be declared as single precision, double precision, integer, or string. The characters **!**, **#**, **%**, and **\$** specify these variable types and are part of the variable's name. Instead of appending the appropriate character to the variables' names, it is also possible to declare variables as integer, single precision, double precision, and string by means of separate statements in the program. Declaring a variable as integer may be useful, since integer values take up less memory than other numeric values. Also, integer arithmetic is faster. In some applications it may be necessary to declare variables as double precision to produce more accurate results.

The statement

*COMMENTS*

```
25 DEFINT A, B, E-G Variables A, B, E, F, G are integers.
```

declares variables **A** and **B**, as well as variables **E** through **G**, as integer variables. These variables and all variables whose names start with these letters

are taken as integers within the program as of line 25. For example, variables ABC, A2, and EZ are then integer variables. However, the variables A2! or EZ# are, respectively, single- and double-precision variables even though their names start with the letters A and E. The type declaration characters (see previous chapter) always override the declaration statement.

The statements

|                  | <i>COMMENTS</i>                            |
|------------------|--------------------------------------------|
| 30 DEFSNG X-Z    | Variables X, Y, Z are single precision.    |
| 35 DEFDBL S, U-W | Variables S, U, V, W are double precision. |
| 40 DEFSTR K, L   | Variables K, L are strings.                |

define the listed variables as single-precision, double-precision and string variables, respectively. These statements may be used in the command mode with and without line numbers.

|                   | <i>COMMENTS</i>                                                       |
|-------------------|-----------------------------------------------------------------------|
| DEFINT I          | The variables are declared.                                           |
| DEFSNG A          |                                                                       |
| DEFDBL D          |                                                                       |
| DEFSTR S          |                                                                       |
| IT=40000          | IT starts with I; overflow occurs since integers cannot exceed 32767. |
| ?OV ERROR         |                                                                       |
| A=1.234567        |                                                                       |
| PRINT A           |                                                                       |
| 1.23457           | In single precision have 6 significant figures.                       |
| D=123456789       |                                                                       |
| PRINT D           |                                                                       |
| 123456789         | In double precision have up to 16 significant figures.                |
| PRINT A+D         | Surprise: A and D can be added.                                       |
| 123456790.2345671 | Result is in double precision.                                        |
| S\$="STRING"      |                                                                       |
| PRINT S\$+A       | Type Mismatch error. Cannot add a string to a number.                 |
| ?TM ERROR         |                                                                       |
| A\$="STRING"      | Declare A\$ as a string variable.                                     |
| PRINT S\$+A\$     | S and A\$ can be concatenated; they are both string variables.        |
| STRINGSTRING      |                                                                       |

**REMEMBER:** The variable identification characters !, #, %, and \$ override the DEFinition statements.

## EXERCISES 4

The following exercises illustrate techniques you have learned in the preceding sections. When you write programs, be sure that the programs actually work properly. Work out some test examples by hand and compare them with the computer's results. Take the time to organize your input and output displays into an easily readable form.

1. Suppose the following program has been entered:

```
10 A=10
20 B=30
30 PRINT "THE PRODUCT OF A AND B IS"; A*B
40 END
```

What display will the following instructions produce

| Instruction   | Anticipated Display | Display |
|---------------|---------------------|---------|
| a. LIST       | _____               | _____   |
| b. LIST 30    | _____               | _____   |
| c. LIST 10-30 | _____               | _____   |
| d. LIST.      | _____               | _____   |
| e. LIST -40   | _____               | _____   |
| f. LIST 30    | _____               | _____   |
| g. EDIT 30    | _____               | _____   |
| h. EDIT.      | _____               | _____   |
| i. EDIT       | _____               | _____   |
| j. RUN        | _____               | _____   |
| k. RUN 20     | _____               | _____   |
| l. 10         | _____               | _____   |
| m. CLS        | _____               | _____   |
| n. RUN        | _____               | _____   |

2. The program listed above needs to be edited to correct the typographical error in line 30. Perform the edit and then list the line to be sure it has been edited properly. Perform the edit by
- Retrying the entire line. List the program to check your editing; then reenter the line with the typographical error in it.
  - Entering EDIT 30 followed by the X command. Move the cursor back, erase the A, and then retype properly.
  - Entering EDIT 30. Then press the space bar up to the D before the A. Then press C followed by U, the proper letter in PRODUCT. Then press ENTER.
3. Enter the following instruction

```
10 PRINT "ABCEF"
```

Edit the line to insert the letter D in the appropriate spot using the I edit command.

4. Edit each of the following BASIC statements. Try several approaches, and for each statement determine the best procedure. First enter the given BASIC statement; then type EDIT N, where N is the line number of the statement to be edited.

**BASIC Statement**

```
10 PRINT
15 INT A
20 PRINT "PHYSICS IS PHUN"
25 PRINT A,B,C
30 INPUT "YOUR AGE"; X
35 END
40 Y=5*(X-32)/9

45 PRINT "DON'T TREAD ON ME"

50 PRINT "ONE FOR THE MONEY"
55 LET X=5
```

**CORRECTION**

```
Delete the extra R
Insert a PU for INPUT
Replace PH by F
Replace the commas by semicolons
Replace the X by AGE
Replace 35 by 99
Replace the Y and X by C and F,
respectively
Remove all characters to the right
of TREAD
Replace the third letter O by $
Delete the LET
```



5. Consider the following program:

```
NEW
10 INPUT A
20 PRINT A
```

The following data are entered for A during execution:

|           | Anticipated Display | Display |
|-----------|---------------------|---------|
| a. 5      | _____               | _____   |
| b. 5, 6   | _____               | _____   |
| c. 4+1    | _____               | _____   |
| d. FIVE   | _____               | _____   |
| e. "FIVE" | _____               | _____   |

6. Consider the following program:

```
10 INPUT A$
20 PRINT A$
```

The following data are entered for A\$ during execution:

|                  | Anticipated Display | Display |
|------------------|---------------------|---------|
| a. FIVE          | _____               | _____   |
| b. 5             | _____               | _____   |
| c. FIVE AND TEN  | _____               | _____   |
| d. 5+10          | _____               | _____   |
| e. FIVE, AND TEN | _____               | _____   |
| f. FIVE:10       | _____               | _____   |
| g. "FIVE:10"     | _____               | _____   |
| h. " TEN"        | _____               | _____   |
| i. TEN           | _____               | _____   |
| j. AB"C          | _____               | _____   |
| k. "FIVE, SIX"   | _____               | _____   |

7. Combine each of the following into a single INPUT statement.

- 10 PRINT "WHAT IS YOUR NAME?"  
20 INPUT A\$
- 10 PRINT "YOUR NAME AND AGE?"  
20 INPUT A\$, AGE

- Modify the mortgage program of the previous section to input with a single INPUT statement the amount of the mortgage, its duration, and the interest rate. Test your modification.
- Telegrams cost \$0.20 per word. Write a program to input the number of words and display the cost.
- Write a program to input three temperatures and compute the average temperature.
  - The temperatures are to be entered one at a time.
  - The temperatures are to be entered all on one line.
 The output is to be of the following form

THE AVERAGE TEMPERATURE IS...

- Degree days are computed by subtracting the day's average temperature from 65. (The average temperature must be less or equal to 65 since negative degree days have

no meaning.) Modify the program of the previous problem to include a display of the day's degree days.

12. Write a program that upon execution yields the following output.

```
HELLO
HI! WHAT IS YOUR NAME?
? you enter your name here
HELLO followed by your name
MY NAME IS NEUTER COMPUTER
```

13. Write a program to input from the keyboard a name, address, city, state, and zip code. Then output these data in the form of an address label. For example,

```
RON DAVIS
382 ORANGE ST
BOSTON, MA 01230
```

Use INPUT for data entry. Be sure to type in NEW before entering the program. If the total number of characters in your address label exceeds 50, type NEW and ENTER, followed by CLEAR 200 and ENTER.

14. Write a program to produce the following dialog:

```
PLEASE ANSWER THE FOLLOWING QUESTION
WHAT DO YOU PREFER: BOYS OR GIRLS?
? enter your choice here...
HEY, I TOO PREFER... THAT'S GREAT!
```

15. Write a program to input two numbers from the keyboard. Then output their sum and difference on one line, and on the following line their product and ratio. Display your results with appropriate descriptive text.
- Input the two numbers on two separate lines.
  - Input the two numbers on the same line.
16. The pressure that a diver experiences as he dives into the ocean is given by the relation

$$P = 0.0295 * H$$

where H is the depth in feet and P the pressure in atmospheres. Write a program to input a depth and compute the corresponding pressure with the following format:

```
AT A DEPTH OF... FEET THE PRESSURE IS... ATMOSPHERES
```

17. If today's population is P people and the population increases each year by I percent, in N years the population will be

$$P * (1 + I/100)^N$$

Write a program to input P, I, and N and then output the following:

```
TODAY'S POPULATION IS...
AT A GROWTH RATE OF ... % PER YEAR
THE POPULATION WILL BE... IN... YEARS
```

# chapter 4 | decisions

The normal sequence of executing statements in a computer program is by increasing line numbers. The statement with the lowest line number is executed first, and the statement with the highest line number is executed last. Thereafter, execution of the program stops. In this chapter we learn techniques that allow programs to deviate from this normal sequence of execution. This process is called **branching** and is made possible by **transfer statements**. As a result, considerable versatility is added to computer programs since they can proceed through different sequences of instructions depending on conditions encountered during execution. For example, if A exceeds B we wish to print A, while if B exceeds A we print B, and if A equals B we print the values of both A and B. So in addition to transfer statements we need to learn about **decision-making** statements. The computer must decide if A is larger than, less than, or equal to B, and execution must then branch to the appropriate print statement. Such decision statements involve relational operations such as greater than and less than.

## 4.1 RELATIONAL AND LOGICAL OPERATIONS

In addition to the arithmetic operations, BASIC also has **relational operations** that are useful for making comparisons. There are six relational operators (same as in Level I):

1. Greater than:  $>$
2. Less than:  $<$
3. Equal:  $=$
4. Greater than or equal:  $>=$  or  $=>$
5. Less than or equal:  $<=$  or  $=<$
6. Not equal:  $<>$  or  $><$

Frequently, rather than determining the value of the larger of two numbers, we may wish to determine whether one number is or is not larger than another number. We may want a yes or a no to the question, is the account overdrawn or is a grade on an exam passing? The result is a logical value, true or false, yes or no. In BASIC, yes corresponds to a -1 and a no to a 0:

|               | COMMENTS                           |
|---------------|------------------------------------|
| PRINT 4>3     |                                    |
| -1            | Yes, 4 is greater than 3.          |
| PRINT 3>4     |                                    |
| 0             | No, 3 is not greater than 4.       |
| PRINT 5=5     |                                    |
| -1            | Yes, 5 equals 5.                   |
| A=2           |                                    |
| B=4           |                                    |
| PRINT B>A     |                                    |
| -1            | Yes, B is greater than A.          |
| PRINT 5*A>B   | 5*A = 10.                          |
| -1            | Yes, 10 is greater than 4.         |
| PRINT 5*(A>B) | A is not greater than B.           |
| 0             | 5 times 0 is zero.                 |
| PRINT 1+B>A   | The sum of 1 and B is greater than |
| -1            | A. The addition is performed first |
|               | and then the comparison is made.   |

**REMEMBER:** A true statement corresponds to a -1  
A false statement corresponds to a 0.

Several expressions containing relational operations can be combined using **logical operations**. There are three logical operators: AND, OR, NOT.

|                        | COMMENTS                             |
|------------------------|--------------------------------------|
| PRINT (3>2) AND (6>-1) | Both expressions are true; the AND   |
| -1                     | yields a true, -1.                   |
| PRINT 3>2 AND (5>10)   | Both expressions are not true; the   |
| 0                      | AND yields a false, 0.               |
| PRINT 3>2 OR 6>-1      | Both expressions are true; the OR    |
| -1                     | yields a true, -1.                   |
| PRINT (3>2) OR (5>10)  | Either expression is true; the OR    |
| -1                     | yields a true, -1.                   |
| PRINT (3>20) OR (5>10) | Neither expression is true; the OR   |
| 0                      | yields a false, 0.                   |
| PRINT NOT (5>3)        | The expression in ( ) is true; the   |
| 0                      | NOT inverts a true to a false.       |
| PRINT NOT 5<3          | The expression is false; the NOT in- |
| -1                     | verts the false to a true.           |
| PRINT NOT (5<3         | The expression is missing a right    |
| ?SN ERROR              | parenthesis.                         |

Expressions involving logical operators need not be placed in parentheses; however, parentheses help make the statements easier to read.

The logical operators AND, OR, and NOT (in Level I BASIC the AND is a \*, the OR is a +, and the NOT is not available) do not perform arithmetic; they perform comparisons and give a true (-1) or false (0) answer. Table 4.1 summarizes their use.

In Table 4.1 reference is made to logical variables A and B. A **logical variable** is a variable that is specified by means of a logical operation.

TABLE 4.1 Logical operations

| Logical Variables |    | AND     | OR     | NOT   |
|-------------------|----|---------|--------|-------|
| A                 | B  | A AND B | A OR B | NOT A |
| -1                | -1 | -1      | -1     | 0     |
| -1                | 0  | 0       | -1     | 0     |
| 0                 | -1 | 0       | -1     | -1    |
| 0                 | 0  | 0       | 0      | -1    |

*COMMENTS*

```

A=5>2
PRINT A A is a logical variable whose value is
-1 -1.
B=-1
PRINT B B is a numerical variable whose
-1 value is -1.

```

Logical operations can only be performed with logical variables, relational expressions, or the numbers 0 and -1. Logical variables always have a value 0 or -1.

*COMMENTS*

```

A=5>2
B=2>5
PRINT A AND B
0

```

A is -1; B is 0.  
A and B are logical variables.  
-1 AND 0 is 0.  
Result as expected.

**REMEMBER:** Logical expressions and logical variables can only have values of 0 and -1.

Logical expressions may also be formed with string variables. When string variables are compared, the letter A is “less” than a B, which is “less” than a C, and so on.

*COMMENTS*

```

PRINT "YES">"NO"
-1
PRINT "YES"="NO"
0
PRINT "YES"<"YESS"
-1
A$="ERIC"
B$="MARION"
PRINT (A$>"BOY") AND (B$>"GIRL")
-1

```

The letter Y is “greater” than the letter N.  
The two words are unequal.  
YES is “less” than YESS.  
Both expressions in the ( ) are true, so the AND yields a -1.

**REMEMBER:** When strings are compared, the letter A is “less” than the letter B, and so on.

TABLE 4.2 Hierarchy of operations<sup>a</sup>

- 
1. Exponentiation:  $X \uparrow Y$  (or  $X[Y]$ )
  2. Negation:  $-X$
  3. Multiplication and division:  $X * Y$  and  $X / Y$  (left to right)
  4. Addition and subtraction:  $X + Y$  and  $X - Y$  (left to right)
  5.  $<$ ,  $>$ ,  $=$ ,  $<=$ ,  $>=$ ,  $<>$  (left to right)
  6. NOT
  7. AND
  8. OR
- 

<sup>a</sup>The innermost parentheses are evaluated first, followed by the next level out. On the same level, operations are performed according to the above order.

The rules for **hierarchy of operations** were introduced earlier. These rules need to be extended at this point to include the relational and logical operators. Table 4.2 summarizes the order in which operations on the same nesting level (for example, within the same parentheses) are performed.

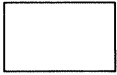
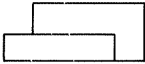
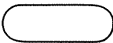

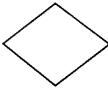
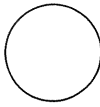
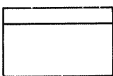
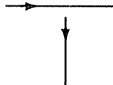
## 4.2 FLOWCHARTS

When faced with the assignment of writing a story, a writer generally first creates an outline. Similarly, when faced with a problem to be solved on the computer, a programmer must first break down the solution process into component steps that can be programmed. It is often convenient to have a pictorial way of displaying the steps to be used in the computer program. This is called a **flowchart**. For complicated problems and complicated computer programs, a flowchart is a must. Even though for less complicated problems a flowchart is not always necessary, it is a good idea to get into the habit of flowcharting each and every program. This approach will help organize your thoughts and avoid major errors in logic. A clear flowchart is then translated into BASIC. The symbols used here for flowcharts are shown in Table 4.3. There is no generally accepted set of symbols for flowcharts. However, these symbols are frequently used.

In the last chapter we developed the compound interest program. A flowchart for that program is given in Figure 4.1. The line numbers of the program corresponding to each step in the flowchart are shown. The beginning of the program is indicated by the START box. The input of the interest rate, initial deposit, and number of years on deposit are contained in input boxes. The balance after N years is then computed and is contained in a rectangular box, which indicates processing of data. The results are subsequently displayed. These are contained in output boxes. The end of the program is shown by the terminal box STOP.

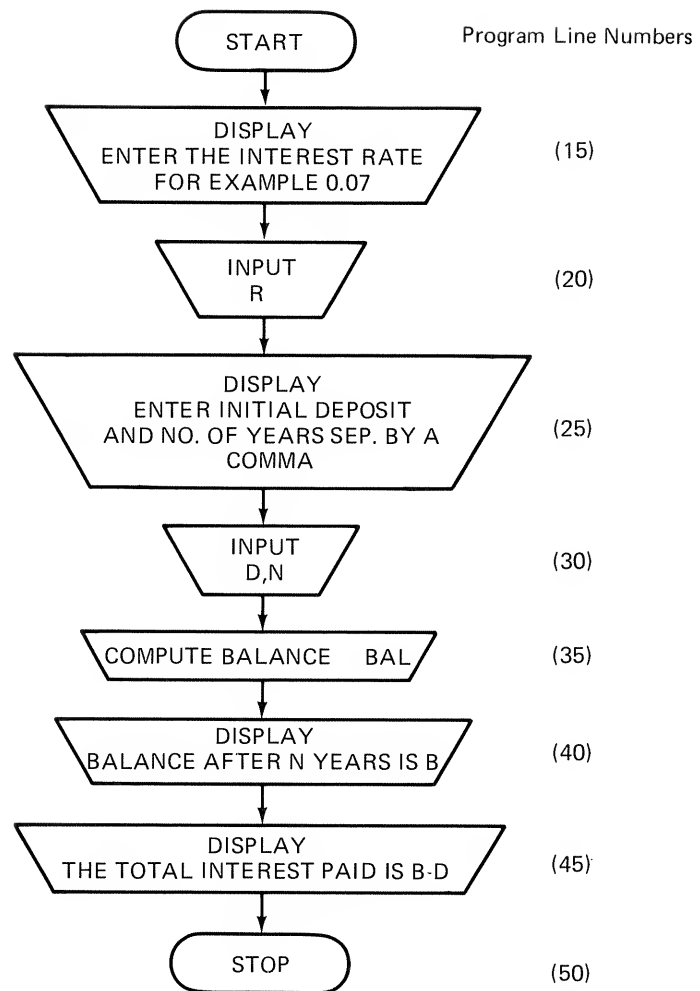
A flowchart to input and compare the values of variables A and B is shown in Figure 4.2. This flowchart illustrates a two-way decision. The variables A and B are first entered and then compared in the decision box. Depending on whether A is or is not greater than B, the appropriate message is displayed. If a program was to be written from this flowchart, the path of its execution would then depend on how large A is with respect to B. However, both paths lead to the same terminal box STOP.

TABLE 4.3 Flowchart symbols

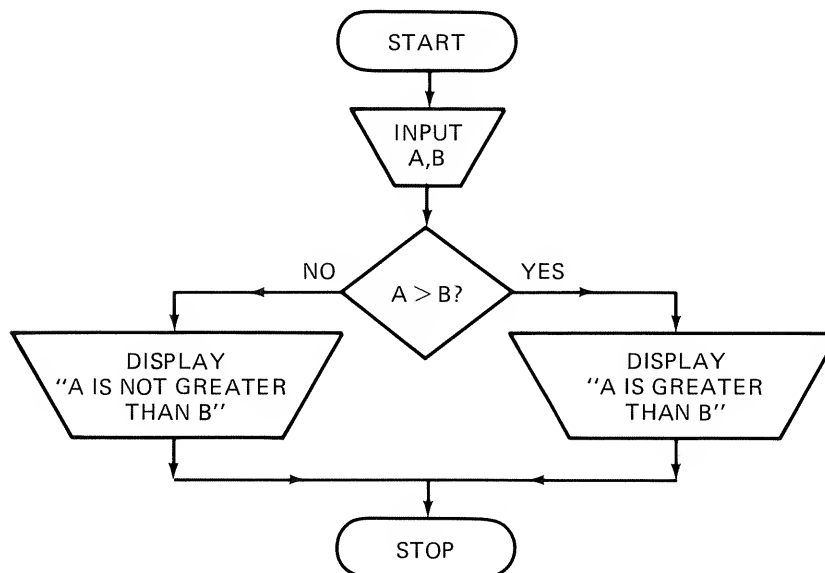
| Symbol                                                                              | Symbol Name       | Type of Instruction                 |
|-------------------------------------------------------------------------------------|-------------------|-------------------------------------|
|    | Processing box    | Processing of data                  |
|    | FOR-NEXT loop box | Sets limits for counter variable    |
|    | Terminal box      | Starting or stopping execution      |
|    | Input/output box  | Input or output of information      |
|    | Decision box      | Decision for conditional transfers  |
|   | Connector         | Connects flowchart segments         |
|  | Subroutine        | Represents a section of the program |
|  | Flow lines        | Indicate direction of program flow  |

The flowchart of Figure 4.2 illustrates a two-way decision. It determines whether *A* is or is not greater than *B*. If *A* is not greater than *B*, it might be important to know whether *A* equals *B* or whether *A* is less than *B*. Figure 4.3 illustrates this three-way decision, which requires two decision boxes. It is an extension of the two-way decision of Figure 4.2.

Flowcharts may have several correct versions. Similarly, there are many ways to write a computer program to perform a specific task. For example, the first decision box in the flowchart of Figure 4.3 could contain the question, is  $A < B$ ? Another possibility is to have the second decision box test if  $A < B$ . In each case the subsequent output boxes would display different messages, but the task would be accomplished equally well.



**FIGURE 4.1** Flowchart of the compound interest program.



**FIGURE 4.2** Flowchart illustrating a two-way decision.



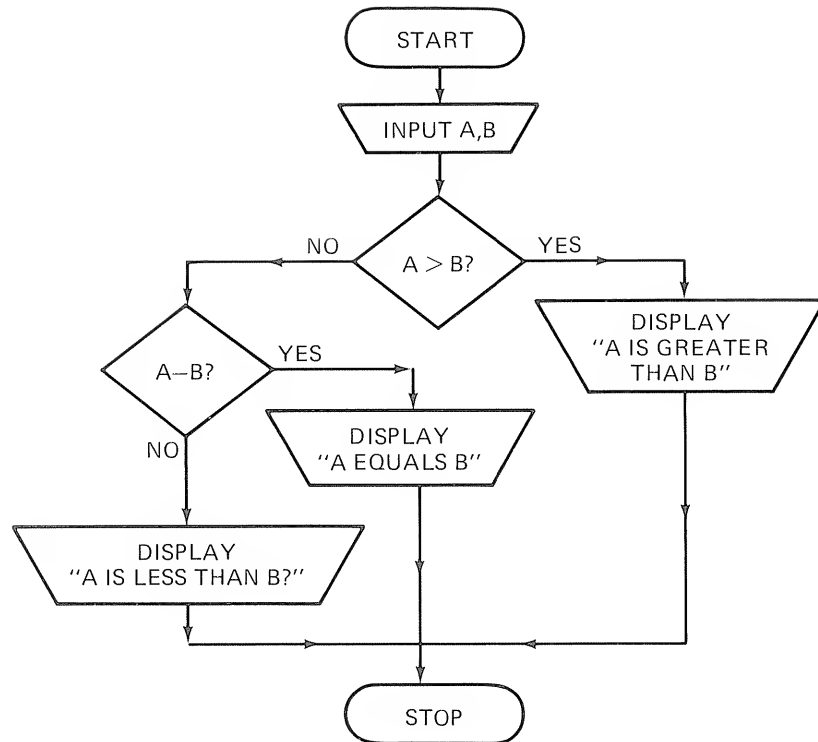


FIGURE 4.3 Flowchart illustrating a three-way decision.

## EXERCISES 5

Before executing the instructions, fill in the anticipated display column and then check against the computer result. When you make a mistake, jot down a short explanation of the mistake.

| 1. | Instruction                           | Anticipated Display | Display |
|----|---------------------------------------|---------------------|---------|
| a. | 20 PRINT 20<br>10 PRINT 10<br>RUN     | _____               | _____   |
| b. | PRINT 3=3                             | _____               | _____   |
| c. | PRINT 5>6                             | _____               | _____   |
| d. | PRINT (5<6)*2                         | _____               | _____   |
| e. | A=10<br>B=30<br>PRINT A=3*B           | _____               | _____   |
| f. | PRINT (3>4) OR (3>1)                  | _____               | _____   |
| g. | PRINT (3>4) AND (3>1)                 | _____               | _____   |
| h. | PRINT NOT (3>4)                       | _____               | _____   |
| i. | PRINT -1 AND -1                       | _____               | _____   |
| j. | PRINT NOT 5                           | _____               | _____   |
| k. | PRINT NOT NOT 0                       | _____               | _____   |
| l. | PRINT (-1>1) AND (-2<10) OR (-1<10)   | _____               | _____   |
| m. | PRINT (-1>1) AND ((-2<10) OR (-1<10)) | _____               | _____   |

```

n. PRINT "JACK" AND "JILL" _____
o. PRINT "JACK" = "JILL" _____
p. PRINT "JACK" > "JILL" _____
q. PRINT "Z">" " _____
r. PRINT "ACE"<"ACE " _____
s. PRINT "ZVIH">"ZWIG" _____
t. PRINT ", "<". " _____

```

2. Draw a flowchart for a program to input two numbers and output their sum.
3. Draw a flowchart for a program to input two numbers and output them in ascending order.
4. Draw a flowchart for a program to input three numbers and output them in ascending order.
5. Draw a flowchart for a program to input three names and output them in alphabetic order.
6. Draw a flowchart for a program to input two pairs of numbers, A, B and C, D. Determine which of the following messages is appropriate; then output that message. We exclude the possibility of  $A = B$  or  $C = D$ .
  - a.  $A > B$  and  $C > D$
  - b.  $A > B$  and  $C < D$
  - c.  $A < B$  and  $C > D$
  - d.  $A < B$  and  $C < D$
7. Draw a flowchart and write the program to input six numbers with a single input statement, and then output two numbers per line.

## 4.3 TRANSFER STATEMENTS

Normally, execution of programs is sequential, starting with the first statement in the program and ending with the last. Transfer statements make it possible to deviate from this normal sequence. There are conditional and unconditional transfer statements.

An **unconditional transfer** causes a change in the order of execution. The instruction

```

.
.
10 GO TO 35
.
.

```

will cause transfer of execution from line 10 directly to line 35, skipping over all intermediate instructions. Other examples of acceptable **GO TO** statements are

```

45 GOTO 75
95 GOTO 15

```

### COMMENTS

From line 45 transfer directly to line 75.

From line 95 transfer back to line 15. The space between the GO and TO is optional.

Commonly the GOTO statement is used without a space.

|                          |                                    |
|--------------------------|------------------------------------|
|                          | <i>COMMENTS</i>                    |
| 10 A=1                   | A sample program to illustrate the |
| 15 B=10                  | GOTO.                              |
| 20 GOTO 40               |                                    |
| 30 PRINT "LINE 30 A="; A |                                    |
| 40 PRINT "LINE 40 B="; B |                                    |
| 50 END                   |                                    |
| <br>RUN                  | Request execution.                 |
| LINE 40 B=10             | Line 30 was skipped over.          |

Note that line 30 will always be skipped over. It is unreachable.

Another use of the GOTO transfer is to cause execution to begin at a specified line number. Recall the RUN statement causes execution to start at the first line of the program. The RUN also sets all numerical variables equal to zero. The GOTO lets you pass values assigned to variables in the immediate mode to variables within a program. We illustrate this feature with the above program.

|               |                                            |
|---------------|--------------------------------------------|
|               | <i>COMMENTS</i>                            |
| B=100         | Specify B = 100.                           |
| GOTO 40       | Start execution at line 40.                |
| LINE 40 B=100 | Result of execution. B is now 100.         |
| <br>RUN       | Request execution.                         |
| LINE 40 B=10  | B is now again 10.                         |
| <br>GOTO 25   | Start execution at line 25.                |
| ?UL ERROR     | Undefined Line error; there is no line 25. |

**REMEMBER:** The line number in the GOTO statement must be an existing line number in the program. It must be a number and cannot be a variable.

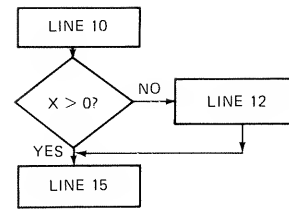
Transfer statements consisting of expressions provide considerably more flexibility. They are called **conditional transfers**. The order of execution is controlled by conditions encountered within the expression of the statement. These transfer statements use the words **IF-THEN-ELSE**. The following examples demonstrate the IF-THEN-ELSE conditional transfers. The same examples are also illustrated along with corresponding flowcharts in Figure 4.4.

|                                      |                                        |
|--------------------------------------|----------------------------------------|
|                                      | <i>COMMENTS</i>                        |
| 10 .....                             |                                        |
| 11 IF X>0 THEN 15                    | If X is positive, transfer to line 15; |
| 12 .....                             | otherwise continue on line 12.         |
| 15 .....                             |                                        |
| <br>20 .....                         |                                        |
| 23 IF A>B PRINT "A IS LARGER THAN B" | If A exceeds B, print message; other-  |
| 25 .....                             | wise continue on line 25 without       |
|                                      | printing.                              |
| <br>30 .....                         |                                        |
| 31 IF A>B AND C>D THEN 38            | If A exceeds B and C exceeds D,        |
| 32 .....                             | transfer to line 38; otherwise con-    |
| 38 .....                             | tinue on line 32.                      |

```

10
11 IF X>0 THEN 15
12
15

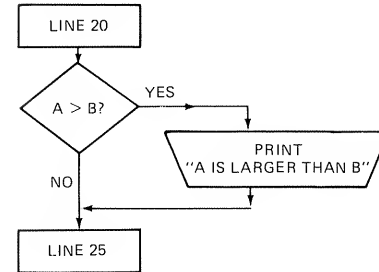
```



```

20
23 IF A>B PRINT "A IS LARGER THAN B"
25

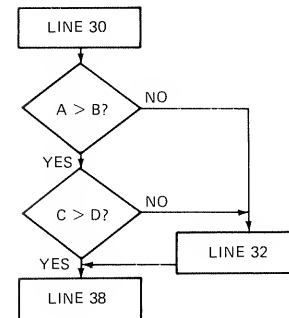
```



```

30
31 IF A>B AND C>D THEN 38
32
38

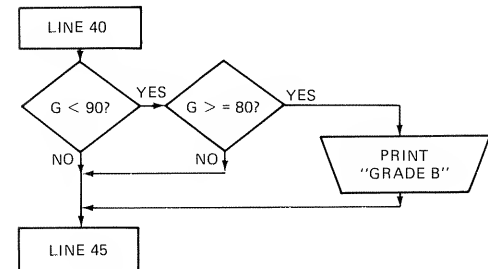
```



```

40
42 IF G<90 AND G>=80 PRINT "GRADE B"
45

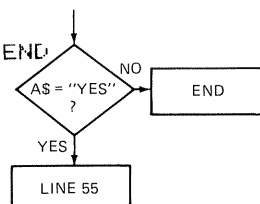
```



```

50 IF A#="YES" THEN 55 ELSE END
55

```



```

60
62 IF A<0 THEN A=-A
65 B=A/5

```

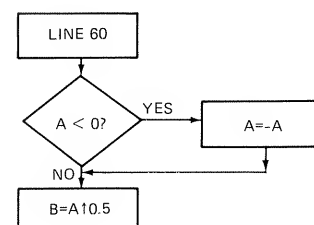


FIGURE 4.4 Flowcharts illustrating IF-THEN-ELSE conditional transfer statements.

|                                      |                                       |
|--------------------------------------|---------------------------------------|
| 40 .....                             |                                       |
| 42 IF G<90 AND G>=80 PRINT "GRADE B" | Only if G is in the 80's, print it as |
| 45 .....                             | a B.                                  |
| 50 IF A\$="YES" THEN 55 ELSE END     | If A\$ is YES, transfer to line 55;   |
| 55 .....                             | otherwise end execution.              |
| 60 .....                             |                                       |
| 62 IF A<0 THEN A=-A                  | If A is negative, respecify it and    |
| 65 B=A <sup>2</sup> .5               | take its square root; otherwise       |
|                                      | take the square root directly.        |
|                                      | Bracket is exponentiation.            |



### Example: Find the Largest Among Three Numbers

We input three numbers and by means of several decisions find the largest. The flowchart for this process is shown in Figure 4.5. It requires three IF-THEN statements to pinpoint the largest number. The line numbers of the corresponding program are shown in the flowchart.

|                                                 |                                    |
|-------------------------------------------------|------------------------------------|
| 10 REM FIND THE LARGEST NUMBER                  | COMMENTS                           |
| 20 PRINT "ENTER 3 UNEQUAL NUMBERS"              | REMinder.                          |
| 30 INPUT A,B,C                                  |                                    |
| 40 IF A>B THEN 70                               |                                    |
| 50 IF B>C THEN PRINT "B IS LARGEST=";B: GOTO 90 | Two statements are chained.        |
| 60 PRINT "C IS LARGEST=";C: END                 | The colon chains them.             |
| 70 IF C>A THEN 60                               |                                    |
| 80 PRINT "A IS LARGEST=";A                      |                                    |
| 90 END                                          | A second END statement.            |
| RUN                                             | Request execution.                 |
| ENTER 3 UNEQUAL NUMBERS                         |                                    |
| ? 1,2,3                                         | The three numbers are entered.     |
| C IS LARGEST= 3                                 |                                    |
| RUN                                             | Request another execution.         |
| ENTER 3 UNEQUAL NUMBERS                         |                                    |
| ? -1,-2,-3                                      | Type in three numbers and press    |
| A IS LARGEST=-1                                 | ENTER.                             |
|                                                 | The minus sign occupies the blank. |

The first decision compares A to B in line 40. If A is larger than B, we transfer to line 70 and compare A to C. If A is also larger than C, we transfer to line 80 where we print the message that A is the largest along with the value of A. Execution is subsequently terminated in line 90.

On the other hand, if A is not larger than B, execution transfers from line 40 to line 50. We then already know that B exceeds A, and now compare B to C. If at this point B is larger than C, then B is indeed the largest and the appropriate message is displayed in line 50. Execution is then terminated through the unconditional GOTO 90. Finally, if in line 60 B is not larger than C, then C must be the largest, since we already established that  $B > A$ . The message "C IS LARGEST" along with its value is displayed in line 60, with execution terminated thereafter.

Aside from illustrating the IF-THEN statement, we introduced two new concepts with the above program: **statement chaining**, resulting in **multiple statement lines**. This is

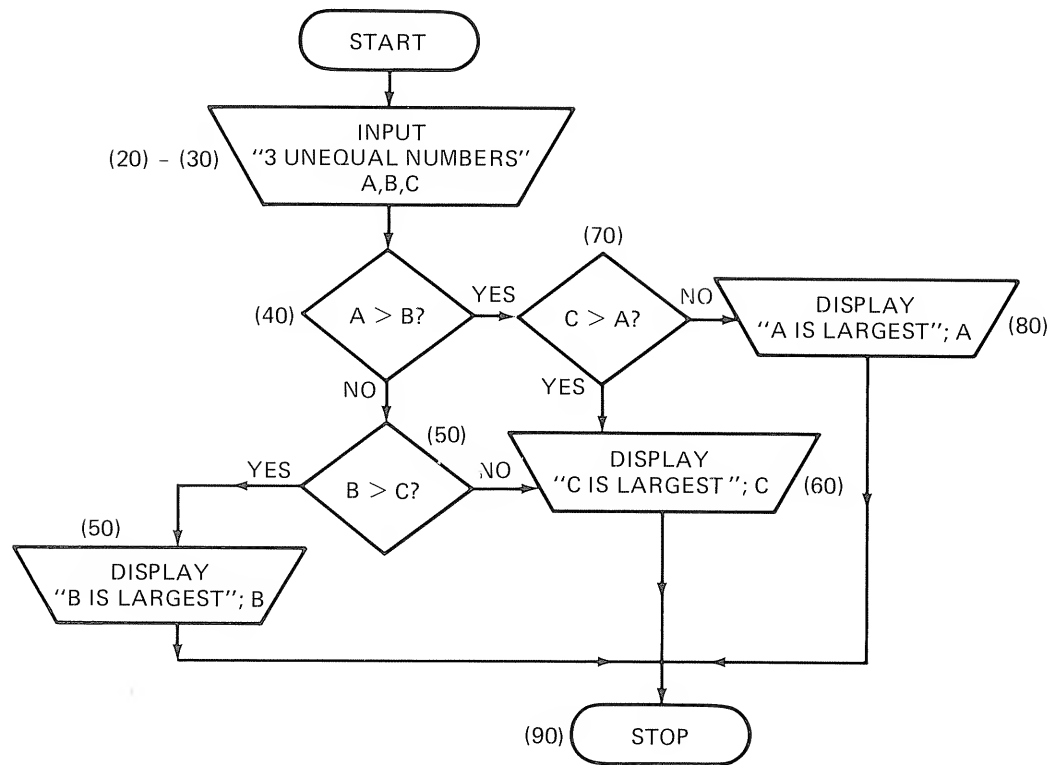


FIGURE 4.5 Find the largest among three numbers.

a useful way of programming since it saves memory space in the computer. You can chain as many BASIC statements as you wish in one line. Chained statements must be separated by colons, and the total number of characters cannot exceed 255.

We also notice that more than one END statement appears in the program. There is an END statement chained in line 60 and an END in line 90. The GOTO 90 in line 50 could also be replaced by an END in order to save the transfer to line 90 from line 50.

**REMEMBER:** Multiple statement lines save memory space but may make it more difficult to read the program.

The IF-THEN-ELSE allows for a two-way decision. Using several IF-THEN statements allows for multiple decisions.

ON expression GOTO 1st line number, 2nd line number, . . .

is a multiway branching statement that causes execution of the program to transfer to the first line number if the value of the expression is 1. If the value of the expression is 2, execution transfers to the second line number, and so on. For example,

```
ON Z GOTO 30, 50, 40
```

will transfer execution to line 30 if  $Z = 1$ , to line 50 if  $Z = 2$ , and to line

40 if  $Z = 3$ . If  $Z$  is 0 or larger than 3, control transfers to the next line in the program after the **ON-GOTO** statement. If  $Z$  is not an integer, for example, 3.5 only the integer portion is considered, in this case 3, and transfer is made to the appropriate line number, in this case line 40. If  $Z$  is negative, an error occurs (FC ERROR). This information is illustrated in the execution of the following program.

```
10 ON Z GO TO 30,50,40
30 PRINT "TRANSFER TO LINE 30 WITH Z=";Z:END
40 PRINT "TRANSFER TO LINE 40 WITH Z=";Z:END
50 PRINT "TRANSFER TO LINE 50 WITH Z=";Z:END
```

#### COMMENTS

|                                                    |                                                                                     |
|----------------------------------------------------|-------------------------------------------------------------------------------------|
| Z=1<br>GOTO 10<br>TRANSFER TO LINE 30 WITH Z=1     | Specify Z.<br>Execute program from line 10.<br>Transferred to first line no.        |
| Z=2<br>GOTO 10<br>TRANSFER TO LINE 50 WITH Z=2     | Transferred to second line no.                                                      |
| Z=3<br>GOTO 10<br>TRANSFER TO LINE 40 WITH Z=3     | Transferred to third line no.                                                       |
| Z=7<br>GOTO 10<br>TRANSFER TO LINE 30 WITH Z=7     | Transferred to first line no., since Z exceeds 3.                                   |
| Z=0<br>GOTO 10<br>TRANSFER TO LINE 30 WITH Z=0     | Transferred to first line no., since Z is zero.                                     |
| Z=1.5<br>GOTO 10<br>TRANSFER TO LINE 30 WITH Z=1.5 | Z is not an integer.<br>Integer portion of Z is 1, so transferred to first line no. |
| Z=-5<br>GOTO 10<br>?FC ERROR IN 10                 | Error, since Z is negative.                                                         |



#### Example: Producing Variable Displays

Write a program to input a person's sex (SEX) (male = 1, female = 3) and his/her age (AGE). Have the computer analyze these data to determine whether the person is or is not a senior citizen ( $\geq 65$  years old).

```
10 INPUT "ENTER PERSON'S SEX (M=1, F=3) AND AGE"; SEX, AGE
20 IF SEX<>1 AND SEX<>3 THEN 10
30 IF AGE<=65 THEN E=SEX:GOTO 50
40 E=SEX+1
50 ON E GOTO 60,70,80,90
60 PRINT "MALE SENIOR CITIZEN":END
70 PRINT "YOUNG MALE":END
80 PRINT "FEMALE SENIOR CITIZEN":END
90 PRINT "YOUNG FEMALE":END
```

This program has some interesting logic. Line 10 is as usual. Line 20 is a **data validity** check. If the person's sex is neither 1 nor 3, execution returns to line 10 where the data need to be entered again. In data-processing applications it is very important that the input be checked. The results are only as good as the input data. A famous saying sums it up best: Garbage In Garbage Out (**GIGO**).

In line 30 of the program, E is equal to SEX if AGE  $\geq$  65. Thus E = 1 for a male and E = 3 for a female senior citizen. If the person's age is less than 65, execution transfers to line 40, where E=SEX+1, that is, E = 2 for males and E = 4 for females who are not senior citizens. So at this point E is either 1, 2, 3, or 4 depending on the person's sex and age. In line 50 we take advantage of this fact and transfer depending on the value of E to line 60, 70, 80, or 90, where the appropriate display is produced.

#### 4.4 ON ERROR GO TO

The data validity check discussed in the previous section is a way of guarding against bad data. Unless detected upon entry, the data would go unnoticed and would be used to generate meaningless results. There are other circumstances where a program seems to be running well, free of syntax and logic errors. Suddenly, after having been used successfully for several runs, an error message occurs and execution is interrupted. For example, the square root of a variable may be computed somewhere within a very long program. If the value of the variable depends on numerous other prior calculations, then it is certainly possible for the variable to be negative on occasion. This results in an error. Another possibility is that a division by zero may occur due to some unforeseen circumstances. Such errors may turn out costly, since they interrupt the execution. After the proper corrections have been made, the entire run needs to be repeated.

The **ON ERROR GOTO** statement sets up an error-trapping routine, which in the event of an error allows the program to continue without a break in execution. The **ON ERROR GOTO** statement must be placed in the program prior to the occurrence of the error that it is supposed to trap.

```

10 ON ERROR GOTO 80
20
30
40 Y=Y2-Y1:X=X2-X1
50 S=Y/X
60 PRINT "THE SLOPE IS"; S
65
70 ON ERROR GOTO 0
75
80 PRINT "WARNING DIVISION BY ZERO; USED 1E-06 INSTEAD OF ZERO"
85 X=1E-06
90 RESUME

```

The **ON ERROR GOTO 0** terminates the error-trapping process. It disables the **ON ERROR GOTO 80** statement of line 10. Consequently, any errors occurring after line 70 will cause a break in the execution without transfer to the error-trapping routine of lines 80 to 90.

The **RESUME** statement located at the end of the **error routine** transfers execution back to the statement in which the error occurred, in this case line



50. The **ON ERROR GOTO** statement with its error routine is generally designed to guard against one specific type of error. However, it will trap all errors, even those it was not designed to trap. The **RESUME** statement can take on several forms:

|                    | COMMENTS                                                    |
|--------------------|-------------------------------------------------------------|
| <b>RESUME</b>      | Resume at line in which error occurred.                     |
| <b>RESUME 0</b>    | Resume at line in which error occurred.                     |
| <b>RESUME 100</b>  | Resume at line 100.                                         |
| <b>RESUME NEXT</b> | Resume at line following statement in which error occurred. |

**REMEMBER:** In the event of an error, the **ON ERROR GOTO -RESUME** prevents a break in execution.

**ERL** is another useful error routine function. It returns the line number at which an error occurred. So if an error occurs at line 20, then **ERL** assumes the value 20. **ERL** equals 0 as long as no error has occurred since the computer has been turned on. In the immediate mode, **ERL** equals 65535 when an error occurs.

|                        | COMMENTS                                                              |
|------------------------|-----------------------------------------------------------------------|
| <b>PRINT ERL</b>       | Computer has just been turned on.                                     |
| <b>0</b>               | <b>ERL</b> equals zero.                                               |
| <b>S=1/A</b>           | Division by zero.                                                     |
| <b>?/0 ERROR</b>       |                                                                       |
| <b>PRINT ERL</b>       | Once an error is made in the immediate mode, <b>ERL</b> equals 65535. |
| <b>65535</b>           |                                                                       |
| <b>10 A+B=C</b>        | Illegal statement.                                                    |
| <b>PRINT ERL</b>       |                                                                       |
| <b>65535</b>           | <b>ERL</b> is 65535 before execution.                                 |
| <b>RUN</b>             | Request execution.                                                    |
| <b>?SN ERROR IN 10</b> | Syntax error in line 10.                                              |
| <b>PRINT ERL</b>       |                                                                       |
| <b>10</b>              | <b>ERL</b> is now 10. Error occurred in line 10.                      |



#### Example: Data Validity Checks

The following portion of a program illustrates the steps required to trap errors made during data entry.

```

100 CLEAR 12
110 ON ERROR GOTO 200
120 INPUT "CUSTOMER'S NAME"; N$
130 INPUT "CUSTOMER'S NUMBER"; A%
140 PRINT "INPUT DATA ARE OK": END
200 IF ERL=130 THEN 230
210 PRINT "NAME MUST BE LESS THAN 13 CHARACTERS"
220 RESUME 120
230 PRINT "CUSTOMER'S ID NUMBER MUST BE LESS THAN 32768"
240 RESUME 130

```

In line 100 we introduce the **CLEAR n** command. Like **CLEAR**, it resets all numeric variables to zero and all string variables to null. In addition **CLEAR n** reserves space for string variables *n* characters in length. *n* may be a number, a variable, or an expression. Once line 100 is executed, the string variable **N\$**, customer's name, can be up to 12 characters long. If a name longer than 12 characters is entered in line 120, transfer is made to the error-trapping routine of lines 200 to 240. In line 130 provisions are made for entering a customer number as an integer. Recall, integer variables can only assume values in the range -32768 to +32767 inclusive. Therefore, if **A%** is larger than 32767 execution will transfer to the error-trapping routine.

The error-trapping routine, lines 200 to 240, displays the appropriate message depending on where the error was made. Variable **ERL** is either 120 or 130, corresponding to errors made in lines 120 or 130, respectively. During execution the error-trapping routine remains unnoticed as long as the data entered conform. In this example the name must be less than 13 characters and the customer's number must be less than 32768. However, when an illegal name is entered, the error routine forces the operator to reenter a proper name. Execution cannot proceed until a proper name has been entered. The same is true for the customer number. The operator is given the opportunity to correct the error made on data entry without a break in execution.

We now execute the program and observe the error-trapping routine.

```

RUN
CUSTOMER'S NAME? ERIC ACE HAROLD
NAME MUST BE LESS THAN 13 CHARACTERS
CUSTOMER'S NAME? ERIC HAROLD
CUSTOMER'S NUMBER? 42345
CUSTOMER'S ID NUMBER MUST BE LESS THAN 32768
CUSTOMER'S NUMBER? 12345
INPUT DATA ARE OK

```

The first customer's name we entered, **ERIC ACE HAROLD**, was too long. The correct name **ERIC HAROLD** was subsequently accepted and the customer's ID number requested. The number 42345 exceeded 32767 and was therefore rejected. Once the proper number, 12345, was entered, the input data were recognized as OK.

---

**REMEMBER:** A string variable can store up to 255 characters only if enough storage space has been **CLEAR**ed.

---



#### *Example: Change for a Dollar Bill*

Suppose a one dollar bill is tendered as payment for a sale of less than \$1.00. Output the change from the sale, assuming that coin collectors have hoarded all the 50¢ coins. A flowchart is shown in Figure 4.6.

The approach taken is to compare the amount of change first with 25. As long as the change **C** exceeds 25, the counter **Q** is incremented to reflect the number of quarters included in the change. When **C** is less than 25, it is compared with 10 to check for dimes. Subsequently, **C** is checked for nickels and pennies. The variables **S**, **C**, **Q**, **D**, **N**, and **P** represent, respectively, the amount of the sale, the change, the number of quarters in the change, and the number of dimes, nickels, and pennies.

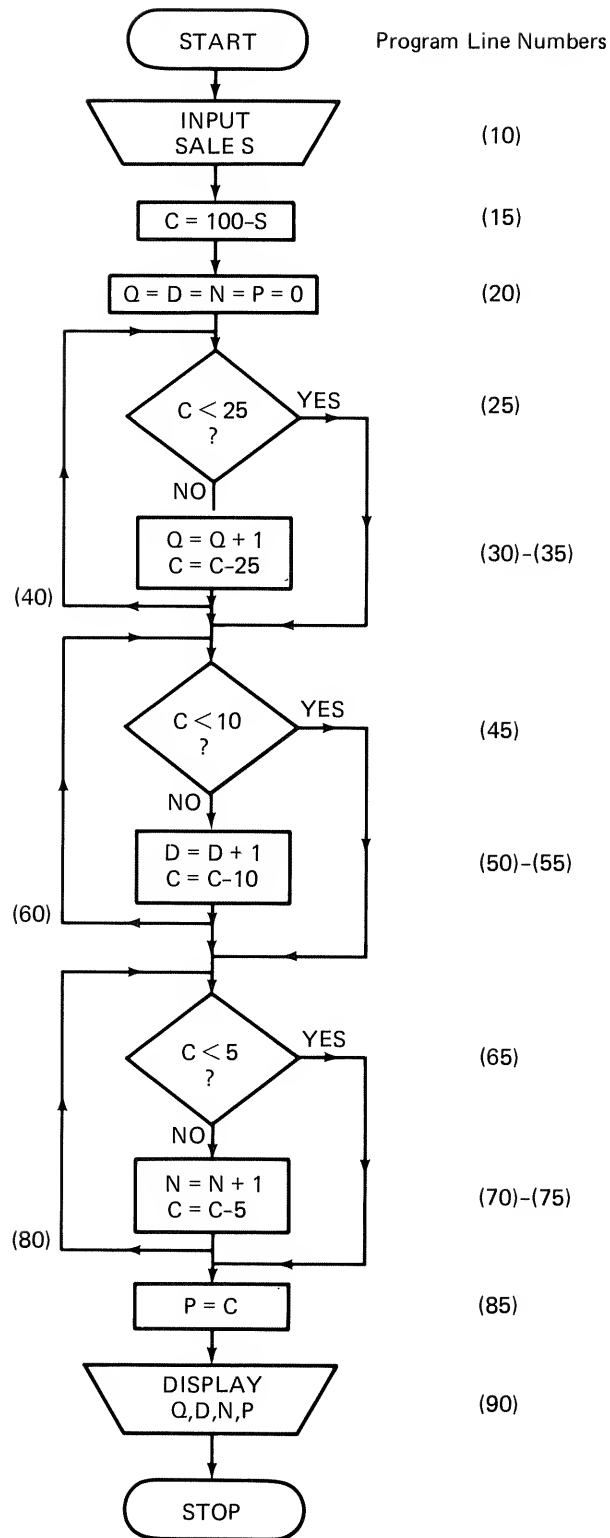


FIGURE 4.6 Change of a dollar bill.

```

10 INPUT "AMOUNT OF THE SALE IN CENTS"; S
15 C=100-S
20 Q=0: D=0: N=0: P=0
25 IF C<25 THEN 45
30 Q=Q+1
35 C=C-25
40 GOTO 25
45 IF C<10 THEN 65
50 D=D+1
55 C=C-10
60 GOTO 45
65 IF C<5 GOTO 85
70 N=N+1
75 C=C-5
80 GOTO 65
85 P=P+1
90 PRINT Q: "QUARTERS" "; D: "DIMS" "; N: "NICKELS" "; P: "PENNIES"

RUN
AMOUNT OF THE SALE IN CENTS? 12
 3 QUARTERS 1 DIMS 0 NICKELS 3 PENNIES

```

In this example we assumed that the amount tendered was 100¢. Can you generalize the program for any amount up to \$100?

A variation of the IF-THEN statement is demonstrated in line 65. Instead of the IF-THEN, it is permissible to use **IF-GOTO** or **IF-THEN GOTO**.

## EXERCISES 6

Before executing the instructions, fill in the anticipated display column and then check against the computer result. When you make a mistake, jot down a short explanation of the mistake.

| 1. | Instruction                         | Anticipated Display | Display |
|----|-------------------------------------|---------------------|---------|
|    | A=10                                | _____               | _____   |
|    | B=20                                | _____               | _____   |
|    | C=-5                                | _____               | _____   |
|    | a. IF A<B PRINT A                   | _____               | _____   |
|    | b. IF A>B PRINT B                   | _____               | _____   |
|    | c. IF A>C PRINT "C"                 | _____               | _____   |
|    | d. IF B>A AND A>C PRINT A*B         | _____               | _____   |
|    | e. IF A>B OR B>C PRINT "YESS"       | _____               | _____   |
|    | f. IF A>B PRINT A ELSE PRINT B      | _____               | _____   |
|    | g. IF A<B PRINT B: PRINT A          | _____               | _____   |
|    | h. IF B>D PRINT "SURPRISE"          | _____               | _____   |
|    | i. IF B>A IF A>C PRINT "UNEXPECTED" | _____               | _____   |
|    | j. PRINT "THE ANSWER IS: PRINT A    | _____               | _____   |

2. Enter the following program:

```

20 IF A$>B$ OR A$>C$ PRINT A$:END
30 IF B$>C$ PRINT B$ ELSE PRINT C$:END

```

| Instruction     | Anticipated Display | Display |
|-----------------|---------------------|---------|
| a. A\$="JOE"    |                     |         |
| B\$="JIM"       |                     |         |
| C\$="JOEY"      |                     |         |
| GOTO 20         |                     |         |
| b. A\$="JILL"   |                     |         |
| GOTO 20         |                     |         |
| c. B\$="MARION" |                     |         |
| C\$="ERIC"      |                     |         |
| GOTO 20         |                     |         |
| d. A\$="ANN"    |                     |         |
| GOTO 20         |                     |         |

3. Write a single statement to
  - a. Set A equal to 5, B to 6, and C to 7.
  - b. Transfer to line 50 if A > B.
  - c. Display C DOES NOT EQUAL D if C is unequal to D.
  - d. Transfer to line 100 if Y is less than Z and W is larger than V.
  - e. Stop execution if Z is 10.
  - f. Branch to line 1 if Z = 1, to line 2 if Z = 2, to line 3 if Z = 3, to line 2 if Z = 4, to line 1 if Z = 5, and to line 4 if Z = 6.
4. Write short programs to perform the following:
  - a. Input two numbers and print their sum only if the sum is positive.
  - b. Input two numbers and print whether their sum or product is greater.
  - c. Display the message, WHAT IS THE CAPITAL OF FRANCE? If the person responds PARIS, have the computer display the message YOU ARE OK. For all other (incorrect) responses, display the message, NOPE, SEE YOU IN PARIS.
  - d. Display the message, WHAT IS YOUR AGE? For ages 12 or less, have the computer respond YOU ARE A CHILD; for ages 13 through 19 respond, YOU ARE A TEENAGER; for ages 20 or above, respond YOU'RE AN ADULT.
5. Draw a flowchart and write a program to input three unequal numbers and display them in ascending order.
6. Draw a flowchart and write a program to input four numbers and display the largest. Try to write the program with the minimum number of steps.
7. Draw a flowchart and write the program to input a number and output its **absolute value**; that is, output the number as a positive number.
8. Draw a flowchart and write a program to compute and display a salesman's gross pay. The pay is based on sales and is computed according to the following formula: for sales up to and including \$1000, he is paid a flat \$100; for sales from \$1000 up to and including \$2000, he is paid 15%; above \$2000, he is paid a bonus of \$250 and 7% of sales. The sales are input to the program.
9. Draw a flowchart and write a program to update the balance of your checking account. Input the amount of the check or the amount of the deposit and output the balance. If the balance falls below \$100, charge the account a \$2 service charge. If the balance drops below zero, display an appropriate message and charge the account a \$5 service charge.
10. Draw a flowchart and write a program to input a person's age and output the age along with the message: THIS PERSON IS A MINOR (<18), or THIS PERSON IS A SENIOR CITIZEN (>=65), or THIS PERSON IS NEITHER A MINOR NOR A SENIOR CITIZEN. Include a **data validity check** to be sure the person's age is not negative or greater than 120.
11. Draw a flowchart and write a program to input the three coefficients A, B, C of the quadratic equation  $AX^2 + BX + C = 0$ . Depending on whether  $B^2 - 4AC$  is posi-

tive, negative, or zero, output respectively the messages: THE ROOTS ARE REAL, THE ROOTS ARE IMAGINARY, THE ROOTS ARE REPEATED.

12. One root of the quadratic equation  $AX^2 + BX + C = 0$  is  $(-B + \sqrt{B^2 - 4AC})/(2A)$ . Write a program to input A, B, and C, and then compute and display the root. Use an ON ERROR GOTO statement to treat the case in which the square root of a negative number is taken, that is, when  $B*B - 4*A*C < 0$ . If  $B*B - 4*A*C$  is positive or zero, display the message "REAL ROOT" along with the root itself. Otherwise display the message "IMAGINARY ROOT". Be sure to first draw a flowchart. Test your program with  $A = 1, B = 3, C = 2$ , and then again with  $A = 1, B = 2, C = 3$ .
13. Write a program to input the number of words in a telegram and display the cost. Assume the first 15 words cost a flat \$3.50, with extra words \$0.20 each.
14. Write a program to input a number and display a message declaring the number as odd or even. Does your program work for positive as well as negative numbers?

# chapter 5 | looping

## 5.1 LOOP STRUCTURE

A major advantage of computers lies in their ability to do repetitive tasks rapidly and accurately. The steps that are repeated within a program form a **loop**. In general, a loop consists of four parts in which we **initialize**, **process**, **increment**, and **test**.

In the initialization part we assign starting values to variables. For example, we may initialize a variable that represents a **counter** to 1. In the process section the calculations are performed. The counter is then incremented, and a test is performed to check if the loop should be continued or terminated. The order in which the four parts of a loop appear is interchangeable. Two possible loop structures are shown in Figure 5.1.

It is important to note that the variables are only initialized once. While looping, the process, increment, and test parts are repeated over and over until the test condition is satisfied, for example, when the counter has reached a preassigned limit. It is also possible to have loops that omit some of these four parts. For example, the increment part may not be necessary if no counter is used. Instead, a computed variable may be checked to terminate the loop.

## 5.2 IF-THEN LOOPS

The test part of the loop can be performed with the **IF-THEN** statement. If the counter has reached its limit, or if a certain condition has been met, the IF-THEN transfers execution out of the loop.

Let us find the sum of all the integers from 1 to 100.

```
10 SUM=0: KOUNT=1
20 SUM=SUM+KOUNT
30 KOUNT=KOUNT+1
40 IF KOUNT<=100 THEN 20

50 PRINT SUM: END

RUN
5050
```

### COMMENTS

Initialize sum and counter.  
Process.  
Increment the counter.  
Test. As long as the counter is less or equal to 100 we continue to loop.

The sum of integers 1 to 100 is 5050.

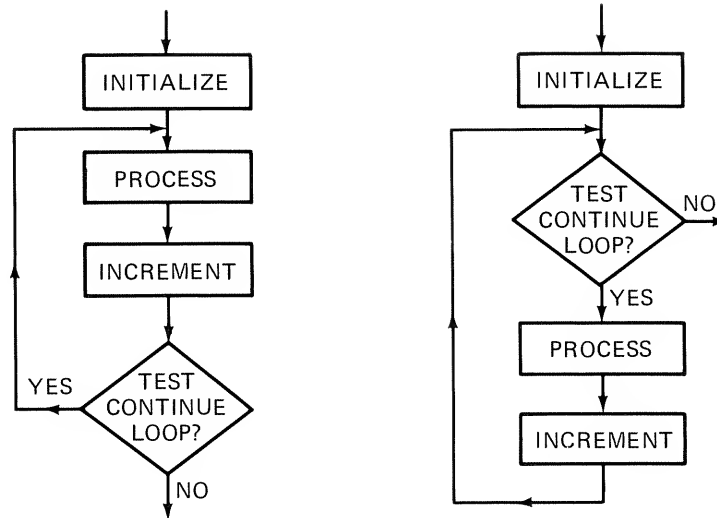


FIGURE 5.1 Two loop structures showing the four parts of a loop.

The next program demonstrates a loop that is terminated when a calculated variable reaches a certain value and not when the counter reaches its limit. Let us find  $N$  such that  $2^N$  just exceeds 1000.

```

10 N=0
20 N=N+1
30 X=2[N
40 IF X<=1000 THEN 20
50 PRINT "N="; N; ", 2[N="; X:END

```

```

RUN
N= 10 , 2[N= 1024

```

## COMMENTS

Initialize.  
Increment.  
Process.  
Test.

We recognize  $2^{10}$  or 1024 to be 1K; a 4K memory thus corresponds to  $4 \times 1024 = 4096$  bytes and not 4000 bytes as is commonly assumed. In the above, lines 30 and 40 can be combined to shorten the program (30 IF  $2[N \leq 1000$  THEN 20). The sequence of the four parts of the loop differs from the illustrations in Figure 5.1.

A common mistake made by inexperienced programmers is to include the initialization statement in the loop. Let us make that mistake by editing line 40 to branch back to line 10 instead of to line 20.

```
40 IF X<=1000 THEN 10
```

```
RUN
```

What happened? The computer is in a coma! No result!? We press BREAK to stop the execution and insert line 35.

```
35 PRINT N
```

```
RUN
```

Now the execution shows only a column of 1's. Press **SHIFT** and **@** simul-



taneously to **halt execution**. The display is frozen. As you can see from the display, the program is in an **infinite loop**. To resume execution, press any key. Since we branch back to line 10, N is reset to zero each time and never increases to a point where  $2^N$  exceeds 1000. Since  $2^1$  is 2, X always equals 2. To get out of this infinite loop, we press the **BREAK** key, which terminates the execution and returns the cursor onto the screen. We also delete line 35, which we inserted for illustrative purposes.

|             | COMMENTS                               |
|-------------|----------------------------------------|
| BREAK IN 30 | Response of computer to BREAK command. |
| READY       | Cursor returned.                       |
| 35          | Delete line 35.                        |
| PRINT X     |                                        |
| 2           | At line 30, $N = 1; 2^N = 2^1 = 2$ .   |

**REMEMBER:** Do not initialize the counter within the loop.

On occasion it may be desirable to have an infinite loop. For continuous loop programs such as games, it is convenient to place the **RUN** statement within the program. If **RUN** is the last statement of the program, then the entire program will be executed over and over. The **RUN** will cause repeated executions and will also set all the variables to zero and null each time an execution starts over.

|                    | COMMENTS                     |
|--------------------|------------------------------|
| 10 PRINT "FOREVER" |                              |
| 20 RUN             | RUN used within the program. |
| RUN                | Request execution.           |
| FOREVER            | First execution.             |
| FOREVER            | Second execution.            |
| FOREVER            |                              |
| :                  |                              |
| etc.               | Infinite loop.               |

Press the **BREAK** key to escape from the infinite loop; to continue execution, type **CONT** and press **ENTER**.



### Example: The Rule of 72

A deposit left in the bank to earn interest will eventually double in value. The **rule of 72** states that the number of years needed for the money to double is approximately equal to 72 divided by the annual interest rate. So, for example, if the interest rate is 4% the time to double will be 18 years ( $72/4$ ). A flowchart for the program to verify this rule is shown in Figure 5.2. The program follows it closely with corresponding line numbers shown in the flowchart.

```

10 REM THE RULE OF 72
20 PRINT "INITIAL DEPOSIT AND INTEREST RATE?"
25 INPUT DEP, IN

```

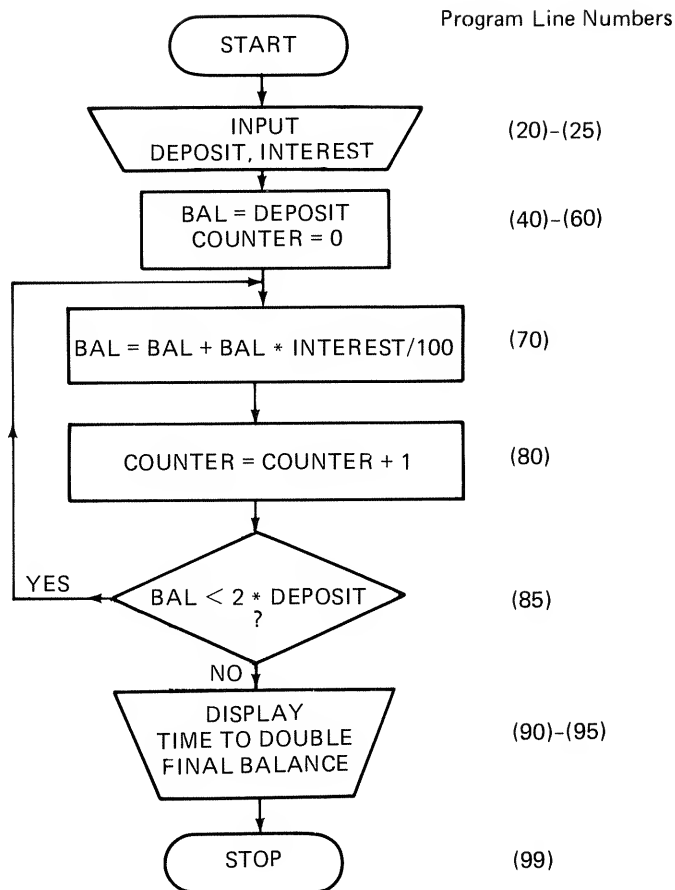


FIGURE 5.2 The rule of 72.

```

30 REM EQUATE THE INITIAL BALANCE TO THE DEPOSIT
40 BAL=DEP
50 REM INITIALIZE A COUNTER
60 N=0
65 REM COMPUTE A NEW BALANCE
70 BAL=BAL+BAL*IN/100
75 REM INCREMENT THE COUNTER
80 N=N+1
85 IF BAL<2*DEP THEN 70
90 PRINT "TIME TO DOUBLE IS";N;"YEARS"
95 PRINT "FINAL BALANCE IS $";BAL
99 END

```

**COMMENTS**

Initialize.

Initialize.

Process.

Increment.

Test.

```

RUN
INITIAL DEPOSIT AND INTEREST RATE?
? 1000.4
TIME TO DOUBLE IS 18 YEARS
FINAL BALANCE IS $ 2025.82

```

Request execution.

Input the data.

```

RUN
INITIAL DEPOSIT AND INTEREST RATE?
? 1000
?? 6
TIME TO DOUBLE IS 12 YEARS
FINAL BALANCE IS $ 2012.2

```

Forgot to enter the interest rate.  
So enter interest rate now.  
Rule of 72:  $72/6 = 12$  years.

The rule of 72 program can readily be used to determine whether a larger (or smaller) initial deposit affects the time to double. Try initial deposits of \$500, \$1000, and \$2000. What is your conclusion? The rule of 72 is verified by means of an exact formula in a later chapter.

---

### 5.3 FOR-NEXT LOOPS

The **FOR** and **NEXT** statements together provide another way of performing loops. The **IF-THEN** loop required a four-part loop structure: initialize, process, increment, and test. The **FOR-NEXT** loop only requires the initialize and process parts. It is generally simpler to use and consequently preferred by many programmers. The general form of the **FOR-NEXT** loop is as follows:

|                                                                     | COMMENTS                                                                                                                              |
|---------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------|
| <pre> 10 FOR A=B TO C STEP D 20 ..... . . 50 NEXT A 60 ..... </pre> | <p>STEP D is optional.</p> <p>The bracket drawn from line 10 to line 50 outlines the range of the loop.</p> <p>The A is optional.</p> |

A is a counter variable whose initial value in the loop is B. The **NEXT** statement is the last statement in the loop. Once that line is reached, execution returns to the beginning of the loop, that is, the **FOR** statement. The counter A is increased by the **STEP D**. If A is less or equal to C, the upper limit of the counter, execution proceeds through the loop. Once line 50 is again reached, execution returns to the beginning of the loop; the counter A is again incremented by D, and checked if it exceeds the value of C. If A does exceed C, the loop has been completed and execution transfers to the first line after the **NEXT** statement, in the above example line 60. If no **STEP** is used, a **STEP** of 1 is assumed. In the **FOR** statement, the initial value, final value, and increment can be constants, variables, or expressions. When the **FOR** statement is reached, these values are evaluated and stored in memory. Respecifying (changing) these values within the loop will not change the range of the loop. However, it is not permissible to change the value of the counter within the loop.

The following examples illustrate the **FOR** statement:

| STATEMENT                     | RANGE OF VARIABLE                                             |
|-------------------------------|---------------------------------------------------------------|
| 1 FOR T = 0 TO 4              | T assumes the values 0, 1, 2, 3, 4.                           |
| 2 FOR F = 6.1 TO 6.4 STEP 0.1 | F assumes the values 6.1, 6.2, 6.3, 6.4.                      |
| 3 FOR Z = 1 TO 10 STEP 2      | Z assumes the values 1, 3, 5, 7, 9.                           |
| 4 FOR M = 10 TO -3 STEP -4    | M assumes the values 10, 6, 2, -2.                            |
| 5 FOR P = 5 TO 3              | P assumes only the value 5.                                   |
| 6 FOR Z=1 TO 8 STEP -1        | Z assumes only the value 1; the spaces are optional in BASIC. |

We notice that the STEP can be positive, negative, integer, or a fraction. In the first example the assumed STEP is 1. In the second example the STEP is a tenth and F is incremented accordingly. In the third example the STEP is 2, and after  $Z = 9$  is incremented by 2,  $Z = 11$ . This is greater than the value 10; therefore, the loop ends without Z ever assuming the final value 10. In the fourth example a negative STEP is demonstrated. Again, the final value is not reached since  $-6$  is less than  $-3$ . In the fifth example no STEP is specified, so STEP 1 is assumed. After P is incremented the first time, its value is 6. Since 6 is greater than the final value 3, the loop ends. In the last example, Z is 0 after it is incremented for the first time. Since 0 is less than 1, the starting value of Z, the loop ends.

**REMEMBER:** If no STEP is specified in a FOR-NEXT loop, a STEP of 1 is assumed by the computer.

The following short program prints the numbers 32 through 34, their squares, and their cubes:

```
10 FOR M=32 TO 34
20 PRINT M; M*M; M^3
30 NEXT
```

```
RUN
32 1024 32768
33 1089 35937
34 1156 39304
PRINT M
35
```

```
10 FOR K=10 TO 5 STEP -2
20 NEXT K
30 PRINT K
```

```
RUN
4
```

#### COMMENTS

STEP 1 is assumed.  
M\*M is computed faster than M^2.  
The M is optional in NEXT.

Upon completion of the loop, the counter has the incremented value that just exceeds the final value;  $M = 34 + 1 = 35$ .

Once the loop is completed the counter is 4;  $K = 6 - 2 = 4$ .



#### Example: Evaluating an Infinite Series

The infinite series

$$1 + 1/2 + 1/3 + 1/4 + \dots + 1/N$$

has an infinite number of terms. Since each successive term is smaller than the previous term, one expects the sum of the series to add up (converge) to a certain value. Does it? Let us investigate the sum of the first 10, 50, and 100 terms, that is, for  $N = 10, 50$ , and 100.

```
20 S=0
30 FOR K=1 TO N
40 S=S+1/K
50 NEXT K
60 PRINT "SUM="; S
```

#### COMMENTS

Initialize the sum S.  
STEP = 1 assumed.

END is optional and we omit it.

We now execute the program by specifying N and then transferring to line 20.

#### COMMENTS

```

N=10
GOTO 20
SUM= 2. 92897 The sum for 10 terms.

N=50
GOTO 20
SUM= 4. 49921 The sum for 50 terms.

N=100
GOTO 20
SUM= 5. 18738 The sum for 100 terms.

```

Indeed this series does not converge; it diverges. Successive terms keep adding on to the total sum, making the sum larger and larger.

In the last example we ran the program for  $N = 100$ . The loop consisting of lines 30 to 50 was executed 100 times as the counter K assumed the values 1 through 100. The sum was then displayed in line 60. Suppose we display the value of the counter upon completion of the loop.

#### COMMENTS

```

N=100
GOTO 20
SUM= 5. 18738
PRINT K The loop's counter is incremented
101 one final time.

```

For a loop ranging from 1 to 20 with a STEP of 2, the counter assumes the values 1, 3, 5 . . . 17, 19. The loop is completed and the counter is incremented one final time. The counter is then 21 and execution of the rest of the program follows.

#### COMMENTS

```

10 FOR K=1 TO 20 STEP 2
20 NEXT K
30 PRINT K

RUN
21

```

The loop's upper limit is 20.  
The STEP is 2.  
The last value of K within the loop is 19.  
Upon completion of the loop, K equals 19 + 2.

If the STEP is 3 in the above program then K is 22 ( $19 + 3$ ) upon completion of the loop.

**REMEMBER:** Every FOR-NEXT loop must have a FOR and a NEXT statement.

FOR-NEXT loops are used frequently and any particular program may contain several such loops. It is possible to branch in and out of loops using conditional or unconditional transfer statements. However, such jumps can be made only within a particular FOR-NEXT loop or from it to the outside. No jump may be made into a loop. A loop may only be “entered” through its FOR statement (first statement of the loop). This means that a GOTO statement, an ON-GOTO statement, or an IF-THEN statement that is outside a FOR-NEXT loop cannot transfer control to any statement inside the loop.

However, a GOTO statement, an ON-GOTO statement, or an IF-THEN statement that is inside a FOR-NEXT loop may transfer control anywhere within or anywhere outside the loop. The following are examples of legal jumps:

|                          | COMMENTS                                                                  |
|--------------------------|---------------------------------------------------------------------------|
| 100 FOR KOUNT=1 TO 100   | Set up a loop.                                                            |
| 150 GOTO 270             | Transfer inside loop.                                                     |
| 270 PRINT "INSIDE LOOP"  |                                                                           |
| 275 IF KOUNT>50 THEN 400 | Transfer inside loop.                                                     |
| 310 IF TEST>90 THEN 500  | Transfer outside loop.                                                    |
| 400 NEXT KOUNT           |                                                                           |
| 560 GOTO 270             | Illegal transfer into loop. A proper version of line 560 is 560 GOTO 100. |



#### Example: Averaging a Set of Numbers

Suppose we know how many numbers we wish to average. Let that number be N. We must add up all the numbers to be averaged and divide the total by N. We first input N and subsequently go through the loop N times to input the numbers to be averaged. In the process we also add up the numbers to compute their sum. Once the loop is completed the average ( $= \text{SUM}/N$ ) is computed and displayed.

```

10 REM AVERAGE A SET OF NUMBERS
20 PRINT "HOW MANY NUMBERS DO YOU WISH TO AVERAGE?"
30 INPUT N
40 REM INITIALIZE THE SUM TO ZERO
45 SUM=0
50 FOR K=1 TO N
60 INPUT "ENTER A NUMBER"; NUMBER
70 SUM=SUM+NUMBER
80 NEXT K
95 PRINT "THE AVERAGE OF THE"; N; "NUMBERS IS"; SUM/N
99 END

```

Type in the program and use it to compute your average grade in mathematics to date, or the average size of the last 12 checks you wrote.

Suppose we wish to average a set of numbers and we do not have an exact count of how many numbers there are. For example, we wish to determine the average amount of a whole pile of checks. We know that the value of a check cannot be a negative number.

```

10 REM AVERAGE A SET OF NUMBERS
15 REM THE NUMBER OF NUMBERS IS UNKNOWN
20 N=0: SUM=0
30 PRINT "ENTER A NUMBER IF DONE ENTER -99"
40 INPUT NUMBER
50 IF NUMBER = -99 THEN 80
60 N=N+1: SUM=SUM+NUMBER
70 GOTO 30
80 PRINT "THE AVERAGE OF"; N; "NUMBERS IS"; SUM/N: END

```

```

RUN
ENTER A NUMBER IF DONE ENTER -99
? 87
ENTER A NUMBER IF DONE ENTER -99
? 95
ENTER A NUMBER IF DONE ENTER -99
? 81
ENTER A NUMBER IF DONE ENTER -99
-99
THE AVERAGE OF 3 NUMBERS IS 87.6667

```

This version of the averaging program uses an IF-THEN loop. The earlier version utilized a FOR-NEXT loop. In line 50 we check the value of NUMBER. If it equals -99 then the last data entry has been made. The -99 serves as a **flag**. Each NUMBER is tested in line 50 to check if it equals the flag. If it does, execution transfers out of the loop to line 80 and the average is displayed.



#### *Example: How Fast Can the Computer Add?*

The program we present in this example can be used to determine the time it takes to perform an addition or any other operation. We begin by executing a loop that actually computes nothing. A stopwatch is used to measure the time to perform the loop 5000 times. The time interval starts when the ENTER key is pressed after typing RUN and ends when the READY appears again.

```

10 FOR S=1 TO 5000
20 A=1
30 NEXT S

```

Execute this program several times and compute the average time of execution. The average time to perform this loop 5000 times turns out to be about 29 seconds. Now replace line 20 by

```
20 A=1+1
```

and run the program again several times. In the presence of the new line 20, it takes about 35 seconds to perform the loop 5000 times. The difference in the two times of execution is 6 seconds. Dividing this difference by 5000 yields the time for a single addition, 0.0012 seconds. Similarly, it is possible to investigate the time of execution for the multiplication  $A = 1*1$ , or compare the difference in the time it takes to execute `PRINT 1` versus `? 1`. Another interesting question this approach may help resolve is whether  $A = 2*2$  is faster or slower than  $A = 2 \uparrow 2$ . Or how about comparing the execution time of the three-line loop (lines 10 to 30) with the execution time of a one-line loop containing the same three statements chained together? Can you think of any other operations that you may wish to investigate?



### Example: The Legend of the Wise Old Man

An old legend has it that a wise man, when offered any reward he desired, requested that the squares of a checkerboard be filled with kernels of wheat. One kernel was to be placed on the first square, two on the second, four on the third, eight on the fourth, and so on. How many kernels of wheat would be required to fill the 64-square checkerboard?

This problem requires summation of 64 integers. Successive integers are in the ratio of 2 : 1. A flowchart is shown in Figure 5.3. The program follows it closely.

#### COMMENTS

```

10 REM THE WISE OLD MAN
20 SUM=0
30 FOR KOUNT=0 TO 63
40 SUM=SUM+2↑KOUNT
50 NEXT
60 PRINT "TOTAL NO. OF KERNELS =", SUM
70 END

```

Initialize the sum;

$2 \uparrow 0 = 1$ ;  $2 \uparrow 1 = 2$ ; etc.  
NEXT by itself is OK.

```

RUN
TOTAL NO. OF KERNELS = 1. 84468E+19

```

Indeed he was a wise man!

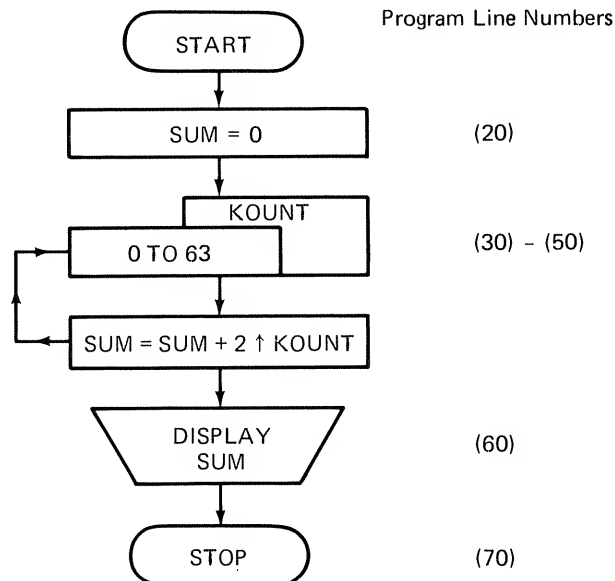


FIGURE 5.3 The wise old man.

## EXERCISES 7

1. What final value will be displayed in each of the following programs:

a. 

```

10 A=4: B=2
20 B=B+A
30 A=A-1
40 IF A>1 THEN 20
50 PRINT B+15

```

b. 

```

10 A=16: B=0
20 B=B*A
30 A=A-2
40 IF A>=0 THEN 20
50 PRINT B+15

```



Anticipated display

---

Actual display

---

c. 10 K=K+1  
20 PRINT K  
30 RUN

d. 10 K=K+1  
20 PRINT K  
30 GOTO 10

Anticipated display

---

Actual display

---

*Note:* Did you remember to type in NEW before entering each program?

2. How often is the IF-THEN executed in each of the following programs? What value will be displayed?

a. 10 A=.5: B=8: C=1  
20 C=C\*(A+7)  
30 B=B-1: A=A+5  
40 IF B>0 THEN 20  
50 PRINT C

b. 10 A=.5: B=1: C=1  
20 C=C\*(A+7)  
30 B=B+1: A=A+5  
40 IF B<=8 THEN 20  
50 PRINT C

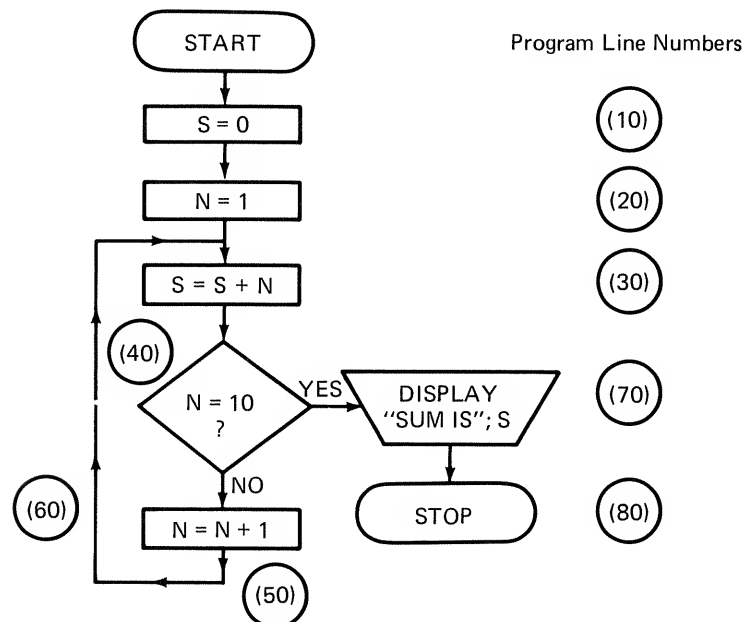
Anticipated display

---

Actual display

---

3. Consider the flowchart for the program to sum the integers (whole numbers) from 1 to 10. The numbers in the circles represent line numbers in the corresponding program.



- Identify the initialization, incrementation, test, and process parts of the loop.
- Write the corresponding program consisting of lines 10 through 80.
- Write a shorter version of the same program.

- d. Generalize the program to sum all the integers from MIN to MAX. Use INPUT to enter values for MIN and MAX during execution.
  - e. Rewrite the program using a FOR-NEXT loop.
4. Write short programs to perform the following. Write these programs with an IF-THEN loop and then with a FOR-NEXT loop. In each case decide which version is the simpler.
    - a. Display all the integers from 1 to 20.
    - b. Display all the odd integers from 1 to 20.
    - c. In increments of 0.5, display all the numbers between 7 and 14.
    - d. Display the fractions  $\frac{1}{3}$ ,  $\frac{1}{4}$ ,  $\frac{1}{5}$ , . . . ,  $\frac{1}{10}$  in decimal form.
    - e. Display every third number starting with 10 and going no lower than -10.
    - f. Determine and display the product of the first 10 positive even numbers.
    - g. Compute and display the product of all the integers from A to B. A and B are input data; assume  $A < B$ .
  5. Modify the rule of 72 program presented in this chapter to display the balance for each year.
  6. Rewrite the rule of 72 program using a FOR-NEXT loop.
  7. Write a program to add the consecutive positive integers and print the last integer added when the sum first exceeds 1000.
  8. A baseball player hits the ball so that the horizontal distance from the plate to the ball in feet is given by the equation  $X = 80T$ , where T is the time in seconds since the ball was hit. The height of the ball above the ground is given by the equation  $Y = 4 + 70T - 16.1T^2$ . Write a program to produce a three-column table giving values of T, X, and Y at  $\frac{1}{2}$ -second intervals as long as the ball has not gone over the fence (X is less than 355 feet).
  9. Write a program that will read 50 scores and then get a count of the number of grades in each of the given ranges 1 to 25, 26 to 50, 51 to 75, and 76 to 100.
  10. You purchase a franchise in a newly formed association and are told that your profit (in \$1000 units) can be projected for the next 8 years by the formula

$$P = T^3 - 5T^2 + 10T - 51$$

where P represents profit and T the time in years. At  $T = 0$ , the time of the purchase of the franchise,  $P = -51$ . Your cost of the franchise is therefore \$51,000. A negative profit indicates a loss. Write a program to determine:

- a. At the close of which year do you show a profit for the first time?
  - b. What is your cumulative total profit, or loss, for the first 8 years?
11. Write a program to compute the net wages of the employees of a company. You are given the gross wages; income tax of 20% if income is \$800 or less, 22% if income is more than \$800; union dues of 1% of total wages; FICA 6% of income equal to or less than \$900. Do not be concerned with dependents. The company has 8 employees.
  12. Write a program to play the following game: The computer tries to guess a number you have in mind from one to 100. First it guesses a number and you tell it if the number is too high, too low, or correct. On the basis of the information you give, the computer guesses again. This continues until the computer guesses right.
  13. Mr. Jones is offered employment by ABC Company, and is afforded the opportunity of taking two different methods of payment. He can receive a monthly payment of \$700 and a \$5 raise each month, or he can receive a monthly wage of \$700 with a yearly raise of \$80. Write a program that will determine the monthly wages for the next eight years in each case. Determine the cumulative wages after each month, and from the information determine which is the better method of payment.

14. How many terms of the series  $1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} \dots$  must be added so that the sum will be greater than or equal to 4?
15. Write a program to input a positive integer  $P$  and display  $P$  as well as  $P!$  (**P factorial**), where  $P! = 1 \times 2 \times 3 \dots P$ ; for example,  $3! = 1 \times 2 \times 3$ .
16. Compute the quantity  $Q = 4 \times (1 - 1/3 + 1/5 - 1/7 + 1/9 - \dots)$ . Input the number of desired terms in the series and output  $Q$ . Do you recognize the quantity  $Q$ ?
17. If the current population is 200,000,000 and the population grows 1.5% per year, find the year in which the population will reach 300,000,000. Use **INPUT** to enter the growth rate and the upper limit on the population.
18. City A has 1000 residents and is agriculturally self-sufficient (i.e., it cultivates enough food to feed itself). In fact, it produces enough food for 100,000 residents. However, every 10 years the population doubles and in that time enough food can be produced to feed 4000 more people than in the previous 10 years. Output a table of data in the following form:

| AFTER YEAR | POPULATION | FOOD SUPPLY FOR |
|------------|------------|-----------------|
| 0          | 1000       | 100000          |
| 10         | 2000       | 104000          |
| .          | .          | .               |
| .          | .          | .               |
| .          | .          | .               |

Have the data stop when the population outgrows the food supply.

19. Job A pays \$12 per day and lasts 30 days. Job B pays \$1 for the first day, \$2 for the second, \$3 for the third, and so on. Which job pays more over a period of 30 days? Obviously, job A is a better deal for a while. When does job B become a better deal?
20. Write a program to compute the balance of a Christmas Club account after one year. The annual interest rate is 6% compounded monthly. Run the program for monthly deposits of \$10, \$20, and \$40.
21. If a person earns 6% compounded monthly on the minimum balance for the month, and he or she withdraws \$100 per month from an account that starts at \$10,000, how long can this go on?
22. Use a computer to help you solve this problem. Mrs. Freezman lives on a street where the house numbers are 1, 2, 3 . . . . It is interesting to note that the sum of the house numbers less than Mrs. Freezman's house number is equal to the sum of the house numbers greater than hers. Her number contains three digits. What is her house number and how many houses are there on the street? It can be shown that the desired house number  $X$  is related to the number of houses on the street  $N$  according to the relation

$$X = -0.5 + \sqrt{0.25 + 0.5N + 0.5N^2}$$

23. An iterative formula for finding the square root of  $N$  is given by the formula

$$B = \frac{G^2 + N}{2G}$$

where  $G$  is the initial guess for the square root of  $N$  and  $B$  is a better approximation.

Count the number of iterations necessary to get successive computed values to differ by no more than  $1\text{E}-6$ . Use  $N = 5$  and  $G = 3$ . (Compute B; then let G be that value of B and compute a new B, and so on.)

## 5.4 SUBSCRIPTED VARIABLES

At times it is desirable to have the computer keep a table of values in its memory, for example, a list of 12 test scores: 65, 63, 75, 76, 90, 83, 87, 98, 76, 75, 81, and 68. The complete set of the quantities is called an **array**, and each individual quantity is called an **element** of the array. The first element of the array is 65, the second element is 63, and the last element is 68. We can give the whole list a variable name, for example, the name S. We then refer to the first item in the list, 65, by S(1). What is S(5)? If  $J = 3$ , what is S(J)? S is a **subscripted variable**; it has one subscript and is sometimes referred to as a one-dimensional array.

Before we can work with our list in a program, we need to store it in the computer's memory. First we must tell the computer to reserve sufficient space for storing our array. We use a **DIM** statement to reserve storage space. For example,

|                  |                                   |
|------------------|-----------------------------------|
|                  | <i>COMMENTS</i>                   |
| 10 DIM S(12)     | Array S is of depth 12.           |
| 20 FOR I=1 TO 12 | S(13) would therefore not be per- |
| 30 INPUT S(I)    | missible.                         |
| 40 NEXT I        |                                   |

Note that in this program the counter of the FOR-NEXT loop is used as the subscript. As I takes on the values 1 through 12, the elements of the array S are entered. Once this program has been executed, the array is stored and we can use its elements; suppose 65 has been entered as the first element and 98 as the eighth element of array S.

|            |                                   |
|------------|-----------------------------------|
|            | <i>COMMENTS</i>                   |
| PRINT S(1) | Display the first element of S.   |
| 65         |                                   |
| PRINT S(8) | Display the eighth element of the |
| 98         | array.                            |

We sum the elements and display their average:

|                               |                                       |
|-------------------------------|---------------------------------------|
|                               | <i>COMMENTS</i>                       |
| 50 SUM=0                      |                                       |
| 52 FOR K=1 TO 12              | Start of loop; the counter is K.      |
| 54 SUM=SUM+S(K)               | The counter is used as the subscript. |
| 56 NEXT K                     | End of loop.                          |
| 58 PRINT "AVERAGE IS"; SUM/12 |                                       |

The following is an example of a DIMension statement

```
10 DIM A(15), B(X), C(D+5)
```

The quantity in ( ) may be a constant, a variable, or an expression. In this case, X and D must be specified prior to line 10.

There are several rules and restrictions we must be aware of:

1. The subscript 0 is permissible. A DIM statement of depth 12, DIM S(12), actually reserves 13 memory locations for S(0), S(1), . . . , S(12).
2. The DIM statement is optional if the largest subscript used for the array is 10.
3. Try to estimate the anticipated size of the array so that excessive memory is not tied up through the DIM statement.
4. The DIM statement may be placed anywhere in the program, but must be placed prior to where the array is first used.
5. The depth specified by a DIM statement may be a number or a numerical expression. DIM S(A + 2) is valid provided A has been specified beforehand.
6. A single DIM statement may be used to declare several variables, for example, DIM A(30), B(40), C(50).
7. A program may contain several DIM statements, but each array can only be declared once.
8. The same variable name can be used in a program for a subscripted variable and for a nonsubscripted variable; S and S(K) are two different variables.
9. The subscript can be an arithmetic expression; S(2 \* 3 + 2) and S(A + 5) are permissible.
10. Arrays may contain numeric or string information. A\$(1) is an array containing strings.



**Example: Find the Largest Element of an Array**

This program demonstrates the use of a subscripted variable, the DIM statement, and two FOR-NEXT loops. A flowchart of the program is shown in Figure 5.4. First the number of elements in the array is entered with an INPUT statement. Once the length of the array is known, the DIM statement is specified as DIM S(N). N is the number of elements in the array. The first loop, lines 30 to 38, is used to input the array S, one element at a time. The array is then examined to locate its largest element. To begin with the first element, S(1) is assumed to be the largest (line 42). It is assigned to variable BIG. Subsequently, the remaining elements S(2) through S(N) are compared with BIG. The FOR-NEXT loop of lines 50 to 58 therefore extends for values of I from 2 to N. Every element S(I) is compared with BIG in line 52. If S(I) exceeds BIG, then an element larger than the current largest element has been encountered. BIG is equated to that element in line 56. Once the loop has been completed the largest element of the array is displayed in line 60.

```

10 INPUT "NUMBER OF ELEMENTS IN THE ARRAY"; N
20 DIM S(N)
30 FOR I=1 TO N
34 INPUT S(I)
38 NEXT I
40 REM ASSUME THE LARGEST ELEMENT IS S(1)
42 BIG=S(1)
50 FOR I=2 TO N
52 IF S(I)<=BIG THEN 58
54 REM S(I) IS LARGER THAN THE PREVIOUS LARGEST ELEMENT
56 BIG=S(I)
58 NEXT I
60 PRINT "THE LARGEST ELEMENT IS"; BIG

```

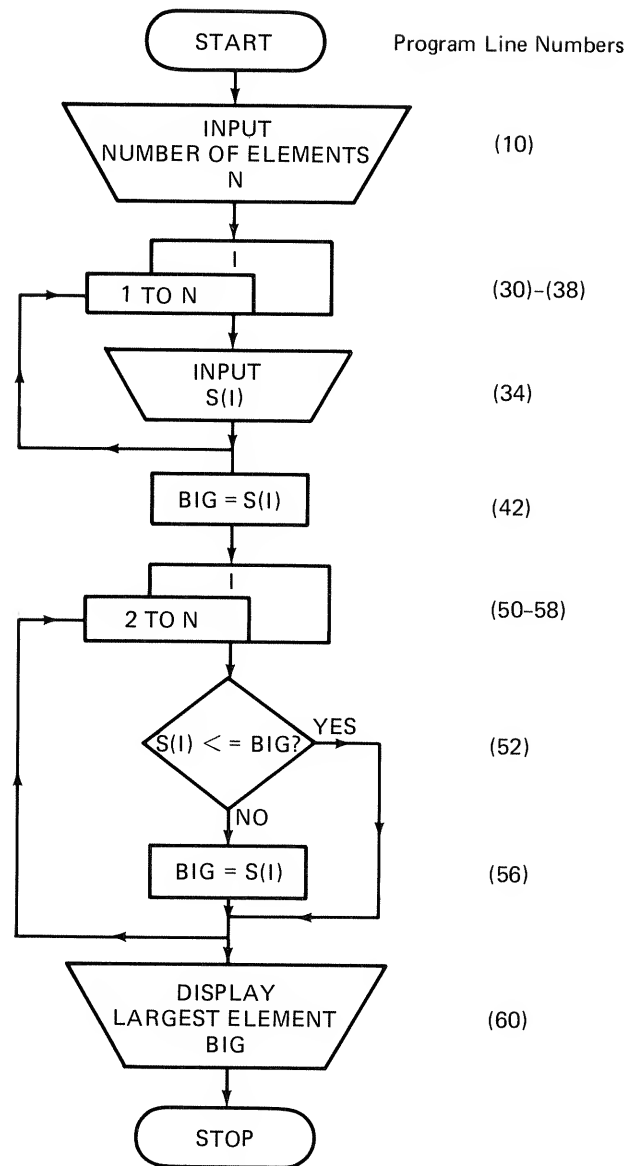


FIGURE 5.4 Find the largest element of an array.

```

RUN
NUMBER OF ELEMENTS IN THE ARRAY? 4
? 10
? 12
? 0
? 34
THE LARGEST ELEMENT IS 34

```

Enter this program and test it. Does it work for negative values as well as for positive values? What if  $N = 1$ ? How would you modify the program to locate the smallest element of array  $S(I)$ ?

Since string data can also be compared and ordered, the above program can be used to find the element in a string array that is closest to the end of the alphabet. Instead of dealing with the numerical array  $S(I)$ , we would deal with the string array  $SS(I)$ .  $SS(I)$  may, for example, contain a list of names.

## 5.5 NESTED LOOPS

If one loop is completely enclosed in another loop, the loops are called **nested loops**. Loops may be nested three, four, or more deep. The only limit is the amount of available memory. Figure 5.5 illustrates the concept of nested loops. It shows a segment of a program that includes several nested loops. The **outer loop** (lines 100 to 650) contains two inner loops. The first inner loop consists of lines 210 to 270. The second inner loop (lines 330 to 590) itself contains an inner loop (lines 400 to 510). The brackets are drawn in Figure 5.5 to outline the range of each loop. Loops are legally nested if the brackets do not intersect.

**REMEMBER:** When you write programs containing nested loops, draw brackets to outline the range of each loop. Be sure the brackets do not intersect.



### Example: The Multiplication Table

We use nested loops to produce a multiplication table for the numbers 4 through 9.

|                             | COMMENTS                   |
|-----------------------------|----------------------------|
| 10 REM MULTIPLICATION TABLE |                            |
| 15 PRINT " 4 5 6 7 8 9"     | Display heading.           |
| 20 FOR OWTER=4 TO 9         | Outer loop starts.         |
| 24 PRINT                    | Skip a line.               |
| 25 PRINT OWTER: " "         | Display numbers at margin. |
| 30 FOR INNER=4 TO 9         | Inner loop starts.         |
| 40 PRINT OWTER*INNER;       | Compute and display.       |
| 50 NEXT INNER               | Inner loop ends.           |
| 60 NEXT OWTER               | Outer loop ends.           |

The index of the outer loop is OWTER. We did not use the variable OUTER since it contains the BASIC reserved word OUT and is therefore an illegal variable. Aside from demonstrating nested loops, this example introduces a new way of tabulating numbers. The **semicolon** at the end of lines 25 and 40 compresses the display on one line. The semicolon at the end of a PRINT statement, sometimes referred to as a **hanging semicolon**, suppresses the line feed. In the absence of line 24, the output would be so compressed that the numbers would fill the entire screen and would make no sense. Line 24 has no hanging semicolon and thus interrupts the compression. Enter the program and run it with and without line 24.

We now execute the program, which includes line 24.

| RUN | 4  | 5  | 6  | 7  | 8  | 9  |
|-----|----|----|----|----|----|----|
| 4   | 16 | 20 | 24 | 28 | 32 | 36 |
| 5   | 20 | 25 | 30 | 35 | 40 | 45 |
| 6   | 24 | 30 | 36 | 42 | 48 | 54 |
| 7   | 28 | 35 | 42 | 49 | 56 | 63 |
| 8   | 32 | 40 | 48 | 56 | 64 | 72 |
| 9   | 36 | 45 | 54 | 63 | 72 | 81 |

The program can be edited to display a multiplication table with a different range,

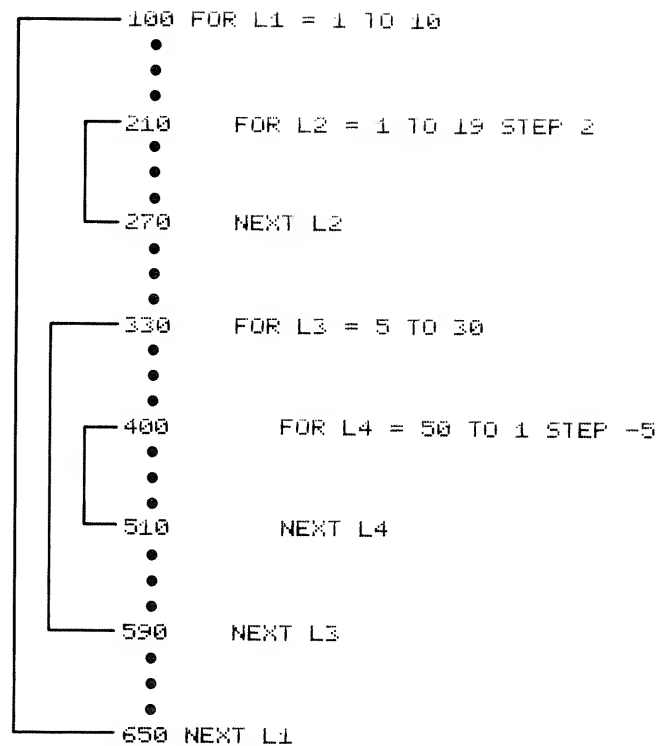


FIGURE 5.5 Nested loops.

for example, for the numbers 1 through 10, or a table for the 10's, 11's, 12's . . . 15's. Which lines need to be edited? The alignment of the numbers in the multiplication table may have to be improved using techniques to be introduced in the next chapter.

The hanging semicolon is further illustrated in the following example:

|                       | <i>COMMENTS</i>                |
|-----------------------|--------------------------------|
| 10 FOR K=1 TO 3       |                                |
| 20 PRINT "ROW";       | Note the hanging semicolon.    |
| 30 NEXT K             |                                |
| 40 PRINT "YOUR BOAT"  |                                |
|                       |                                |
| RUN                   |                                |
| ROW ROW ROW YOUR BOAT | Entire display is on one line. |

Now delete the semicolon in line 20 and again RUN the program.

|                | <i>COMMENTS</i>                      |
|----------------|--------------------------------------|
| 20 PRINT "ROW" | No hanging semicolon.                |
|                |                                      |
| RUN            |                                      |
| ROW            | Display appears on successive lines. |
| ROW            |                                      |
| ROW            |                                      |
| YOUR BOAT      |                                      |



## 5.6 MULTIPLE SUBSCRIPTS

Subscripts are used to identify individual numbers in a list of numbers called an array. **AGE (2)** and **AGE (N)** refer to the second and Nth item in the array called **AGE**. Such arrays are known as singly subscripted arrays. They are one-dimensional arrays.

It is also possible to use arrays with more than one subscript. When two subscripts are used, the array is called a **matrix**. It is a two-dimensional array. The elements in a matrix are arranged in rows and columns. Consider the problem of storing the ages of students in a classroom. The classroom has three rows of chairs with each row containing five seats. A matrix containing the desired information is shown below.

| Row | Column |    |    |    |    |
|-----|--------|----|----|----|----|
|     | 1      | 2  | 3  | 4  | 5  |
| 1   | 21     | 19 | 20 | 18 | 17 |
| 2   | 20     | 28 | 18 | 16 | 19 |
| 3   | 19     | 20 | 19 | 18 | 22 |

The matrix has 3 rows and 5 columns. The student in row 2 and column 3 is 18 years old. This group of ages can be stored in the memory of the computer in a doubly subscripted array. The programmer must give it a name, following the same rules used for naming any variable. The array could be named **AGE**. It must then be decided which of the two subscripts in **AGE (J,K)** is to refer to the row and which to the column. Frequently, the first subscript is taken to represent the row number and the second to represent the column number. The memory location **AGE (2,3)** would then represent the age of the student sitting in row 2 and column 3. **AGE (2,3)** is 18. Space for these numbers is reserved in memory by the **DIM** statement

**DIM AGE (3,5)**

This statement will cause the computer to reserve space for 15 numbers. The general form of our matrix is **AGE (ROW, COL)**. The **DIM** statement is optional if both subscripts do not exceed 10. For the array **AGE**, no dimension statement is necessary up to **AGE (10, 10)**. Since **AGE (0,0)** is permissible, we can have up to an 11 × 11 matrix in memory without a **DIM** statement. All the rules mentioned in describing arrays with single subscripts can be extended to variables with multiple subscripts.

The idea of a double-subscripted array can be extended to multiple subscripts. For example, the array **AGE (ROW, COL, CLASS)** is a triple-subscripted array in which the three subscripts represent the row number, column number, and the class that the individual is attending. **AGE (2,3,4)** refers to the person's age who sits in row number 2, column 3, and attends class number 4.



### *Example: Magic Squares*

Consider the following matrix of numbers:

|   |   |   |
|---|---|---|
| 2 | 9 | 4 |
| 7 | 5 | 3 |
| 6 | 1 | 8 |

This matrix is a magic square. It has the property that the sum of the numbers along each row, column, and diagonal is the same. We now write a program to input these numbers into a two-dimensional array SQ, and then check if it is indeed a magic square.

```

10 REM MAGIC SQUARE
15 FOR ROW=1 TO 3
20 PRINT "ENTER 1 NO. AT A TIME FOR ROW
 NUMBER"; ROW
25 FOR COL=1 TO 3
30 INPUT SQ(ROW, COL)
35 NEXT COL, ROW
40 REM CALC. & PRINT SUM OF EACH ROW
45 FOR ROW=1 TO 3
50 SUM(ROW)=0
55 FOR COL=1 TO 3
60 SUM(ROW)=SUM(ROW)+SQ(ROW, COL)
65 NEXT COL
70 PRINT "THE SUM OF ROW"; ROW; "IS"; SUM(ROW)
75 NEXT ROW
80 REM CALC. & PRINT SUM OF A DIAGONAL
85 SUM=0
88 FOR L=1 TO 3
90 SUM=SUM+SQ(L, L)
92 NEXT L
95 PRINT "SUM OF DIAGONAL"; SUM
99 END

```

```

RUN
ENTER 1 NO. AT A TIME FOR ROW NUMBER 1
? 2
? 9
? 4
ENTER 1 NO. AT A TIME FOR ROW NUMBER 2
? 7
? 5
? 3
ENTER 1 NO. AT A TIME FOR ROW NUMBER 3
? 6
? 1
? 8
THE SUM OF ROW 1 IS 15
THE SUM OF ROW 2 IS 15
THE SUM OF ROW 3 IS 15
SUM OF DIAGONAL 15

```

This program is a little longer than it needs to be. The loop to sum the diagonal elements (lines 88-92) can be combined into the nested loops that compute the sum of the elements in each row (lines 45-75). Can you do it?

A new statement is introduced in line 35: **NEXT COL, ROW** terminates two loops. The inner loop has the first counter variable, COL, and the outer loop's counter variable is ROW.

#### COMMENTS

No DIM needed since ROW and COL are assumed less than 11. This line is split to keep program and comments separate.

Enter 1 element of SQ at a time. Both loops end here.

Initialize a sum for each row.

Initialize SUM. SUM is a different variable from SUM(ROW). SQ(L,L) is a diagonal term; SQ(2,2) is the diagonal term in the second row.

---

## 5.7 DEBUGGING LOOPS: TRACING AND PLAYING COMPUTER

In this section we present two techniques most helpful in debugging loops: playing computer and tracing a loop. In the process of illustrating these two

methods of debugging, we will consider errors that are commonly made in programs containing loops. These errors are not syntax errors, but rather errors in logic that result in meaningless answers or infinite loops.

The **trace** function lets you check on the execution of a program. As the execution proceeds from line to line, each and every line number is displayed. These line numbers appear inside brackets and represent sequentially all the lines that were executed, even repeatedly within a loop. Once the program is entered, type in **TRON**, and press ENTER followed by **RUN** and ENTER. To remove the trace, type **TROFF**. Once the trace is completed, we examine the flow of the execution for any irregularities. The trace is done for us by the computer. On the other hand, **playing computer** requires a manual trace. The programmer goes through the program manually line by line, carrying out its logic and computations in an attempt to find the instructions that are responsible for the logic error. Frequently, the trace and playing computer are used together to debug a program.



#### *Example: Sorting a List of Numbers in Descending Order*

The problem we consider in this section is that of **sorting** a list of numbers in descending order, from the high to the low value, and displaying the result. We use the subscripted variable  $S(I)$  where  $I = 1$  to 10. The array  $S(I)$  therefore contains 10 numbers. This program is an extension of the program to determine the largest element of an array, which was presented in this chapter. Once the largest element is found it must be placed in  $S(1)$ . Subsequently, elements  $S(2)$  through  $S(10)$  need to be checked to again locate the largest element among them. And so on until the last element,  $S(10)$ , is reached. It is necessarily the smallest element of the original array  $S(I)$ .

We enter the following program:

```

10 REM SORT IN DESCENDING ORDER
20 INPUT "LENGTH OF ARRAY"; N
30 PRINT "ENTER THE ELEMENTS ONE AT A TIME"
32 FOR I=1 TO N
34 INPUT S(I)
38 NEXT I
40 FOR K=1 TO N-1
42 REM ASSUME S(K) IS LARGEST
50 FOR I=K+1 TO N
52 IF S(I)>S(K) THEN 59
54 REM PERFORM AN INTERCHANGE
56 S(K)=S(I)
59 NEXT I,K
61 PRINT "THE SORTED ARRAY"
62 FOR L=1 TO N: PRINT S(L): NEXT
64 END

```

#### *COMMENTS*

Begin search for Kth largest element.  
Search begins with element  $K + 1$ .  
Located an element larger than  $S(K)$ ?  
Place the larger element in  $S(K)$ .  
End of inner and outer loops.  
  
Entire loop in one line of code.

We now check the program with test data. We will sort the two-element array 10, 20. The expected result is the sorted list 20, 10.

```

RUN
LENGTH OF ARRAY? 2
ENTER THE ELEMENTS ONE AT A TIME
? 10
? 20

```

#### *COMMENTS*

You enter 2 for N.  
  
 $S(1) = 10$ .  
 $S(2) = 20$ .

```

THE SORTED ARRAY
10
20

```

Oops!  
It is not in descending order.

The program needs to be debugged. We will trace the program. There is no need to trace the beginning of the program where the array is entered. So we put a trace on the section of the program in which the sort takes place, between lines 40 and 59.

```

39 TRON
60 TROFF

```

The **TRON** and **TROFF** are now part of the program. Once the program is debugged, these two lines will have to be deleted.

```

RUN
LENGTH OF ARRAY? 2
ENTER THE ELEMENTS ONE AT A TIME
? 10
? 20
(40)<(42)<(50)<(52)<(59)<(60)THE SORTED ARRAY
10
20

```

#### COMMENTS

You enter 2 for N.

S(1) = 10.

S(2) = 20.

Sequence in which lines were executed.

Array not sorted properly.

We now examine the trace. In line 40 the outer loop is set up with  $K = 1$ ; line 42 is only a REM. In line 50 the inner loop is started with  $I = 2$ . In line 52 the comparison  $S(2) < S(1)$  is made. But  $S(2)$  is not  $< S(1)$ , so next we expect to branch to line 56. The trace however indicates that line 59 is executed after the test of line 52. There is a bug here. Looking over line 52 we notice that the subscript J is an error; it should be S(I). We edit line 52 and execute the program.

```

52 IF S(I)<=S(K) THEN 59
RUN
LENGTH OF ARRAY? 2
ENTER THE ELEMENTS ONE AT A TIME
? 10
? 20
(40)<(42)<(50)<(52)<(54)<(56)<(59)<(60)THE
 SORTED ARRAY
20
20

```

#### COMMENTS

Edit line 52: Replace S(J) by S(I).  
Request execution.  
You enter 2 for N.

S(1) = 10.

S(2) = 20.

The trace. Line is split to separate program and comments.

OK, 20 is the larger number.

There's a bug here. This should be a 10.

We still have a bug. The first element of the sorted array is indeed 20, but the second is supposed to be 10. We now play computer as we go through the trace.

After the data have been entered, the loop on K is set up in line 40. The following

table is the result of playing computer:

| Line No. from Trace | K     | I           | S(1) | S(2) | Comments                                         |
|---------------------|-------|-------------|------|------|--------------------------------------------------|
| 40                  | 1     |             | 10   | 20   |                                                  |
| 42                  | REM   | Unchanged   |      |      |                                                  |
| 50                  | 1     | $K + 1 = 2$ | 10   | 20   |                                                  |
| 52                  | 1     | 2           | 10   | 20   | S(2) is not $\leq$ S(1);<br>transfer to line 54. |
| 54                  | REM   | Unchanged   |      |      |                                                  |
| 56                  | 1     | 2           | 20   | 20   | S(1) = S(2).                                     |
| 59                  | 1     | 2           | 20   | 20   | Inner and outer loops<br>exhausted.              |
| 60                  | TROFF |             |      |      |                                                  |

The trace and playing computer verifies the result of the sorted array having the elements 20 and 20. Where is the bug? In looking over the table we notice that both elements become equal to 20 in line 56. The difficulty must be there. The bug is in the way we interchange elements. What we want is for S(K) to be what S(I) was and for S(I) to be what S(K) was. The values of the two variables are interchanged as follows:

#### COMMENTS

```
55 TEMP=S(K)
56 S(K)=S(I)
57 S(I)=TEMP
```

TEMP is a temporary variable.  
This line we already have.  
S(I) is specified = TEMP = S(K).

```
RUN
LENGTH OF ARRAY? 2
ENTER THE ELEMENTS ONE AT A TIME
? 10
? 20
(40)(42)(50)(52)(54)(55)(56)(57)(59)(60)THE SORTED ARRAY
20
10
```

Bravo! We now remove the trace by deleting lines 39 and 60. Type 39 and ENTER. Type 60 and ENTER.

```
RUN
LENGTH OF ARRAY? 3
ENTER THE ELEMENTS ONE AT A TIME
? 10
? -5
? 20
THE SORTED ARRAY
20
10
-5
```

The program now works. We found two bugs, which resulted in the following three corrections to the original program:

1. Replace the J by an I in line 52
2. Add the line: 55 TEMP = S(K)
3. Add the line: 57 S(I) = TEMP

Another approach to debugging, which is not demonstrated in the example, is to introduce several **PRINT** statements into the program. These display intermediate values of variables that may help identify irregularities.

It is also possible to debug a loop by introducing several **STOP** statements into the program. Execution will **BREAK** at each **STOP** with a line number message. The values of the variables of interest can then be printed and examined in the immediate mode. To continue the execution, type **CONT** and press **ENTER**.

**REMEMBER:** Every program, however short or “simple,” needs to be tested. Use test data for which the answers are known. If the results are not as expected, trace the program and play computer to determine where the bug is creeping in.

## EXERCISES 8

- Before executing the instructions shown, fill in what you anticipate the display will be. Add an explanation for any errors you make.

|        |         |                 |
|--------|---------|-----------------|
| Assume | A(1)=2  | A\$="YES"       |
|        | A(2)=20 | A\$(5)="NO"     |
|        | B(1)=6  | C(2,5)=7        |
|        |         | B\$(3,7)=" SIR" |

| Instruction           | Anticipated Display | Display |
|-----------------------|---------------------|---------|
| a. PRINT A(1)         | _____               | _____   |
| b. PRINT A(A(1))      | _____               | _____   |
| c. PRINT A(1)+B(1)    | _____               | _____   |
| d. PRINT A(3)         | _____               | _____   |
| e. PRINT B(11)        | _____               | _____   |
| f. PRINT A            | _____               | _____   |
| g. PRINT A\$          | _____               | _____   |
| h. PRINT A\$+A\$(5)   | _____               | _____   |
| i. PRINT C(2,5)       | _____               | _____   |
| j. PRINT C(2,11)      | _____               | _____   |
| k. PRINT A\$+B\$(3,7) | _____               | _____   |

- Enter the following program, but *do not run* it.

```

10 DIM A(40)
20 FOR L=0 TO 30
30 A(L)=L
40 NEXT L

```

Now enter the instructions shown below in the order given, and fill in what you anticipate the display will be. Add an explanation for any errors you make.

| Instruction | Anticipated Display | Display |
|-------------|---------------------|---------|
| PRINT A(20) | _____               | _____   |
| PRINT L     | _____               | _____   |
| RUN         | _____               | _____   |
| PRINT A(0)  | _____               | _____   |
| PRINT A(20) | _____               | _____   |
| PRINT A(35) | _____               | _____   |
| PRINT A(-1) | _____               | _____   |
| PRINT A(41) | _____               | _____   |
| PRINT L     | _____               | _____   |

3. What integer numbers would be stored in each element of array M by the following program?

```

10 A=1
20 FOR I=1 TO 3
30 FOR L=1 TO 2
40 M(I,L)=A
50 A=A+2
60 NEXT L, I

```

4. Write a program to display the integers 1 through 6.
- One integer per line.
  - All the integers on one line.
  - Three integers per line.
  - Repeat parts a, b, and c for the integers 1 through M; input M.
5. What integers would be stored in each element of the array by the following program?

```

10 A=13
20 FOR B=1 TO 3
30 FOR C=1 TO 2
40 FOR D=1 TO 2
50 E(B,D,C)=A
60 A=A-1
70 NEXT D, C, B

```

6. Investigate how much memory is taken up by each of the following statements. Recall, variable MEM contains that information.
- DIM A(10)
  - DIM A(100)
  - FOR X=1 TO 10: A(X)=X: NEXT X
  - Any other instructions you might be interested in.
7. Write a program to set the even-numbered elements of array A to 5 and the odd-numbered elements to 10. The array has 20 elements.
8. Write a program to set each element of array M to its element number. M is a one-dimensional array of length 30.
9. The program below computes the total number of items in an inventory. Enter the program and run it.

```

10 DIM N(4), P(4), Q(4)
11 REM RESERVE SPACE FOR 4 ITEMS, 4 PRICES AND 4 QUANTITIES
99 REM HERE IS THE POINT WHERE INPUT BEGINS
100 FOR I=1 TO 4
110 INPUT "ITEM CODE NUMBER"; N(I)
120 INPUT "ITEM PRICE $"; P(I)
130 INPUT "QUANTITY IN STOCK"; Q(I)
140 NEXT I
200 REM THIS PART OF THE PROGRAM DISPLAYS INFORMATION
210 C=0
220 FOR K=1 TO 4
230 PRINT "THERE ARE"; Q(K); "OF ITEM NUMBER"; N(K)
240 C=C+Q(K)
250 NEXT K
300 REM WE WILL PRINT THE TOTAL CARRIED AS "C"
310 PRINT "THE TOTAL NUMBER OF ITEMS WE HAVE FOR SALE IS"; C

```

Modify the program to compute and display the following:

- a. Total value of the inventory.
- b. Average cost of an inventory item.

10. Input a list of ten numbers and display the list in reverse order.
11. Given a positive number A and an integer M, compute  $A^M$  without using  $\uparrow$ .
12. Write a program to input a list of five names and corresponding ages. Display the names and ages of those persons whose age is over 65. Use the following data:

MORRIS 75, JOAN 43, SANDY 46, SARA 68, PAULA 16

13. Write a program to display the names and ages of the previous problem in ascending order on age.
14. Write a program to input five first names and output (a) the name that is alphabetically closest to the beginning of the alphabet, and (b) closest to the end of the alphabet.
15. Modify the magic square program to compute the sum of the elements in each column. Then check if all the sums are equal and display the appropriate message, THE SUMS OF ALL THE COLUMNS ARE (ARE NOT) EQUAL.
16. The balances in a checking account at the end of each of the first six months of the year were 230, 450, 610, 610, 380, 420. If the balance on December 31 was 710, write a program to input the data, and create an array giving the net amounts deposited or withdrawn during each of the six months.
17. A company handles 12 items. Initially, there are 1000 of each item. Write a program using subscripted variables that will assign a price to each item. Have the program input two numbers N and C, and have the computer display the cost of item N, and also tell how many items will be left in storage after C of these have been sold.
18. Two matrices A and B are each of dimension  $M \times N$ , where  $M = 3$  and  $N = 2$ . Write a program that compares the corresponding elements of A and B and then forms a matrix that contains the larger of the elements. Have the computer display the matrix of larger elements.
19. A special event was attended by 18 people. They paid a total of \$24.60 for admission. The men paid \$2.21 per ticket, the women paid \$1.77 per ticket, and the children paid 99¢. How many children attended the event? *Hint*: Use three **nested loops**.
20. Write a program that inputs a two-dimensional list of numbers and displays the row number(s) and column number(s) of those rows and columns whose elements are all larger than 100. Use the following array to test the program:

|     |     |     |     |
|-----|-----|-----|-----|
| 100 | 195 | 182 | 225 |
| 85  | 125 | 235 | 67  |
| 129 | 342 | 100 | 750 |

21. Given a ticket number, check the list of ten lucky numbers to see if the ticket is a winner. Display an appropriate message.
22. Given a list of numbers sorted in ascending order, **merge** a number X into the list. Display the new list.
23. The **mode** of a list of numbers is the value that occurs most often. An array may have more than one mode. Write a program to accept an array and display the mode(s) and the number of times the mode(s) occur.



24. **Merge** two lists of numbers, each sorted in descending order, into a single array also sorted in descending order. Display the new array. Modify the program to handle string variables.
25. In football, points are scored as follows:
- |                                             |          |
|---------------------------------------------|----------|
| A touchdown with a successful conversion    | 7 points |
| A touchdown without a successful conversion | 6 points |
| A field goal                                | 3 points |
| A safety                                    | 2 points |
- Write a program to input a final score and output the number of combinations of 2's, 3's, 6's and 7's that will produce that score. For example, 9 points can be scored by the combinations  $2 + 2 + 2 + 3$ ,  $3 + 3 + 3$ ,  $6 + 3$ , and  $7 + 2$ . The order in which the points are scored is not important; that is, the combinations  $7 + 2$  and  $2 + 7$  are the same. The output should be of the form:

... POINTS CAN BE SCORED IN ... DIFFERENT WAYS.

26. Consider nine slips of paper numbered with the digits 1, 2, 3, ..., 9. Group the nine slips into three three-digit numbers, such as 467, 291, and 538. Determine the grouping that will give the smallest possible product of the three three-digit numbers. *Hint:* One number is in the 100's, the second in the 200's, and the third in the 300's.

# chapter 6 | input-output

Input-output statements communicate information to and from the computer. We have used the INPUT statement to enter data from the keyboard during execution. This approach gives the user of the program considerable flexibility, since he can use different data each and every time the program is executed. This advantage is however offset by the fact that a program containing INPUT statements can only operate with human intervention and can therefore not be run unattended until all the interactive dialog has been completed. For some data entry, the INPUT is quite awkward. For example, large arrays are typed in one element at a time, a very slow and tedious process. In this chapter we present the READ and DATA statements, as well as INPUT # for data entry. The READ statement accesses data stored in the program; the INPUT # reads data stored on tape.

The PRINT statement has been used to produce displays on the screen. We have however been unable to control precisely where on the screen the output appears and also the way in which the output is displayed. For example, we may wish to output a neat financial report with the \$ signs positioned in the first position preceding the numbers. The PRINT USING statement specifies the format of the output.

When a program generates output that serves as input for another program, it is convenient to output these data onto tape. The data can then be read from the tape at a later time. The PRINT # is used for that purpose.

## 6.1 READ AND DATA STATEMENTS

The READ and DATA statements together permit reading data. The READ specifies the variables' names whose numeric or string values are located in the DATA statement. The general form of the READ statement is

READ list of variables separated by commas

and the general form of the DATA statement is

DATA list of values separated by commas

Here are some examples of READ and DATA statements:

| EXAMPLES                                        | COMMENTS                                                                                                                           |
|-------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------|
| 10 READ A, B, C<br>15 DATA 1, 2, 3              | This is equivalent to the single statement: 10 A = 1: B = 2: C = 3                                                                 |
| 20 DATA 0, 1, "GIGO"<br>25 READ SUM, KOUNT, A\$ | For every variable there is a corresponding value in DATA. DATA can appear before READ. Here SUM = 0, KOUNT = 1, and A\$ = "GIGO". |
| 30 READ A, B<br>35 DATA 1, 2, 3                 | The DATA statement may have more values than needed. The extras are ignored.                                                       |
| 40 READ A, B<br>44 READ C<br>48 DATA 1, 2, 3    | Each READ statement continues reading the DATA statement where the previous READ left off. Here A = 1, B = 2, and C = 3.           |
| 50 READ A, B, C<br>54 DATA 1, 2<br>56 DATA 3    | Successive DATA statements are like one long line of data. Here A = 1, B = 2, and C = 3.                                           |
| 60 READ A, B, C<br>65 DATA 1, 2                 | Results in an OD error (Out of Data). No value furnished for C.                                                                    |

The statement

```
1 READ X
```

tells the computer to find the first DATA statement in the program and take the value of X from the list of values in the DATA statement. Each time a READ statement is executed, the computer reads the next value from the DATA statement. The computer remembers what values have already been read, that is, where it left off with the last READ statement. If there are no more values left in the DATA statement, the computer moves on to the next DATA statement. The computer considers all DATA statements as one long line of data. If there are no more DATA statements, an OD error (Out of Data) occurs.

**REMEMBER:** For every variable appearing in a READ statement, there must be a corresponding value in a DATA statement. DATA statements can appear anywhere in the program.

The DATA statement can contain strings or numeric constants. No expressions are allowed. The strings do not have to be enclosed in quotes unless they include commas, colons, or leading blanks.

| COMMENTS                                                                     |
|------------------------------------------------------------------------------|
| 10 READ A\$, B\$<br>15 DATA JOHN SMITH, "SMITH, JOHN"                        |
| The quotes around SMITH, JOHN are required because of the comma in the name. |

The *RESTORE* statement causes the next *READ* statement to start over with the first item in the first *DATA* statement of the program. This makes it possible to reuse the same data within a program.

#### COMMENTS

|                                                        |                                                                                                                             |
|--------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------|
| <pre>10 READ A,B:DATA 1,2,3 20 RESTORE 30 READ C</pre> | <p>The <i>DATA</i> statement can be chained. <i>A</i> = 1, <i>B</i> = 2, <i>C</i> = 1 and not =3 due to <i>RESTORE</i>.</p> |
|--------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------|



#### Example: Unit Pricing

The unit price of an item is computed by dividing its price by its weight. The result is the price of the item per unit of weight. For example, an item weighing 50 grams and costing 75 cents has a unit price of 1.5¢ per gram. The program illustrates this computation for three items.

```
10 FOR K=1 TO 3
20 READ W(K),P(K)
30 PRINT "ITEM NO";K;"UNIT PRICE IS";P(K)/W(K);"CENTS PER GRAM"
40 NEXT K
50 DATA 215, 94, 521, 265, 405, 179

RUN
ITEM NO 1 UNIT PRICE IS .437209 CENTS PER GRAM
ITEM NO 2 UNIT PRICE IS .508637 CENTS PER GRAM
ITEM NO 3 UNIT PRICE IS .441975 CENTS PER GRAM
```

This example shows that *READ-DATA* can also be used in conjunction with subscripted variables. In addition, the program may be run for another group of three items by retyping line 50 followed by *RUN*; for example,

```
50 DATA 5, 16, 10, 31, 8, 28

RUN
ITEM NO 1 UNIT PRICE IS 3.2 CENTS PER GRAM
ITEM NO 2 UNIT PRICE IS 3.1 CENTS PER GRAM
ITEM NO 3 UNIT PRICE IS 3.5 CENTS PER GRAM
```

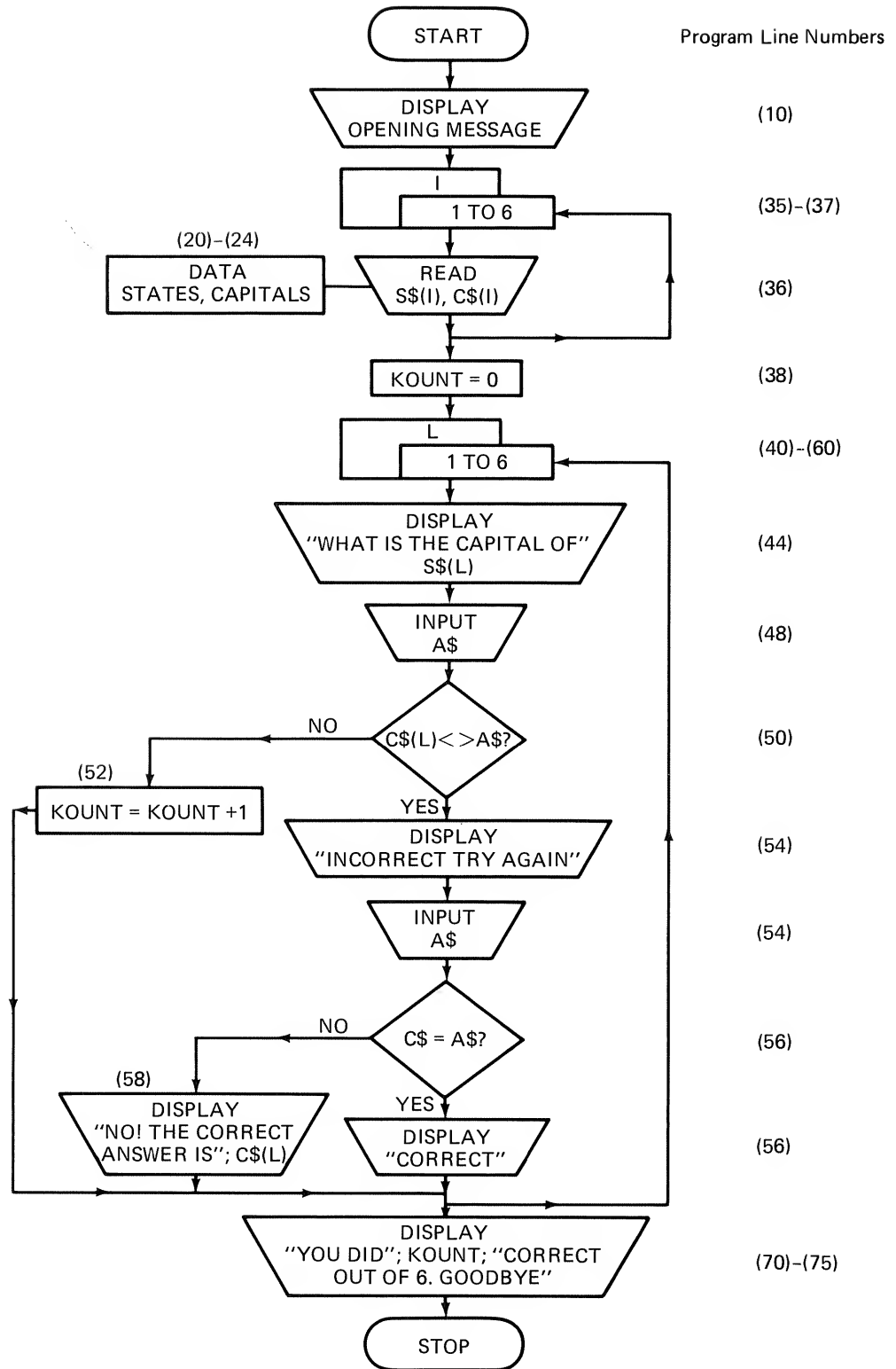
**REMEMBER:** The *DATA* statement provides a means of data entry at the time of execution. Edit the *DATA* statement and then *RUN*.

The above program computes three unit prices. This is limiting. Can you generalize the program so that any number of unit prices can be computed?



#### Example: Practice Your State Capitals

The following program helps you learn the capitals of the New England states. The states and their respective capitals are coded into the *DATA* statement. They are part of the program. A flowchart for the program is shown in Figure 6.1.



**FIGURE 6.1** Practice your state capitals.

```

10 CLS: PRINT "DO YOU KNOW THE CAPITALS OF THE SIX NEW
 ENGLAND STATES?"
20 DATA CONNECTICUT, HARTFORD, MAINE, AUGUSTA
22 DATA MASSACHUSETTS, BOSTON, NEW HAMPSHIRE, CONCORD
24 DATA RHODE ISLAND, PROVIDENCE, VERMONT, MONTPELIER
30 REM. READ IN STATES S$ AND RESPECTIVE CAPITALS C$
35 FOR I=1 TO 6
36 READ S$(I), C$(I)
37 NEXT I
38 KOUNT=0: REM BEGIN QUIZ
40 FOR L=1 TO 6
44 PRINT "WHAT IS THE CAPITAL OF "; S$(L); "?"
48 INPUT A$
50 IF C$(L) <> A$ THEN 54
52 KOUNT=KOUNT+1: GOTO 60
54 PRINT "NOPE, TRY AGAIN": INPUT A$
56 IF C$(L)=A$ PRINT "NOW YOU HAVE IT!": GOTO 60
58 PRINT "NO, THE CORRECT ANSWER IS "; C$(L)
60 NEXT L
70 PRINT: PRINT "YOU GOT"; KOUNT; "OUT OF 6"
75 PRINT "IT WAS A PLEASURE SERVING YOU": END

```

```

RUN
DO YOU KNOW THE CAPITALS OF THE SIX NEW ENGLAND STATES?
WHAT IS THE CAPITAL OF CONNECTICUT?
? HARTFORD
WHAT IS THE CAPITAL OF MAINE?
? AUGUSTA
WHAT IS THE CAPITAL OF MASSACHUSETTS?
? BOSTON
WHAT IS THE CAPITAL OF NEW HAMPSHIRE?
? MANCHESTER
NOPE, TRY AGAIN
? DONT KNOW
NO, THE CORRECT ANSWER IS CONCORD
WHAT IS THE CAPITAL OF RHODE ISLAND?
? PROVIDENCE
WHAT IS THE CAPITAL OF VERMONT?
? BURLINGTON
NOPE, TRY AGAIN
? MONTPELIER
NOW YOU HAVE IT

YOU GOT 4 OUT OF 6
IT WAS A PLEASURE SERVING YOU

```

This program can be executed over and over again without ever having to reenter the names of the states and their capitals. The INPUT statement would not be appropriate in this case because the student may not know the capitals himself. If the teacher entered them for him, it would be extremely inconvenient to reenter them each and every time. A possible alternative to the READ-DATA statements is in this case a series of LET statements setting separately each state and each capital to S\$(I) and C\$(I).

---

## 6.2 FORMATTING OUTPUT

Instructions on how to display the items of the PRINT list are called **formatting** instructions. They determine the format of the display. So far we have formatted our output with semicolons and blanks placed between the

items to be printed. For example,

```
PRINT 1; 2; " " ; -3
1 2 -3
```

With the semicolon separating the numbers, each positive number is printed with a leading and a trailing blank. For negative numbers the leading blank is taken up by the minus sign. There are therefore two spaces between the 1 and the 2; the string of three blanks between the 2 and the -3 results in four spaces in the output. It is often desirable to tabulate the output. This means specifying precisely the column in which the item in the PRINT list will be displayed. A **comma** placed between each item to be printed ensures that successive items start 16 spaces apart. Up to four numbers per line can thus be printed. The screen has four **zones**. So using commas gives a zoned format.

```
PRINT "ZONE 1", "ZONE 2", "ZONE 3", "ZONE 4"
ZONE 1 ZONE 2 ZONE 3 ZONE 4
```

If more than four items are printed, the output takes up more than one line. Successive commas make it possible to skip zones.

```
PRINT 1, 2, 3, 4, 5, 6
1 2 3 4
5 6

PRINT 1, , 2, , 3, , 4
1 2
3 4
```

Normally each PRINT statement begins a new output line. Output from two or more consecutive PRINT statements can be made to appear on the same line by placing a comma after the last item in the PRINT statement. This **hanging comma** will cause the next PRINT statement to display on the same line. For example,

```
10 PRINT "JACK",
20 PRINT "AND",
30 PRINT "JILL"

RUN
JACK AND JILL
```

This is in contrast to **hanging semicolons**, which were introduced earlier.

```
10 PRINT "JACK";
20 PRINT "AND";
30 PRINT "JILL"

RUN
JACKANDJILL
```



### Example: A Sales Report

A car agency with three salesmen sells two car models, A and B. The commission on the sale of model A cars is \$125; commission on the sale of model B cars is \$95 for the first

10 cars and \$145 for subsequent sales. The sales for each model by each of the three salesmen are as follows:

|         | Salesman 1 | Salesman 2 | Salesman 3 |
|---------|------------|------------|------------|
| Model A | 6          | 9          | 4          |
| Model B | 8          | 5          | 12         |

The following program uses READ-DATA statements to input the data and then outputs a sales report. The sales data are stored in three one-dimensional arrays: N\$(I) contains the salesmen's names; ASALES(I) and BSALES(I) refer to sales of models A and B, respectively. The subscript I takes on the values 1, 2, and 3 for the three salesmen.

#### COMMENTS

```

10 REM SALES REPORT
12 FOR I=1 TO 3
15 READ N$(I), ASALES(I), BSALES(I)
18 DATA ERIC, 6, 8, RON, 9, 5, JEFF, 4, 12
21 ACOM(I)=125 * ASALES(I)
24 CHECK=BSALES(I)-10
27 IF CHECK>0 THEN 36
30 BCOM(I)=95 * BSALES(I)
33 GOTO 39
36 BCOM(I)=950+145 * CHECK
39 NEXT I
42 REM PREPARE REPORT
45 CLS: PRINT "SUPER AGENCY SALES REPORT": PRINT
48 PRINT , N$(1), N$(2), N$(3)
51 PRINT "MODEL A SALES", ASALES(1), ASALES(2), ASALES(3)
54 PRINT " COMMISSIONS", ACOM(1), ACOM(2), ACOM(3)
57 PRINT "MODEL B SALES", BSALES(1), BSALES(2), BSALES(3)
60 PRINT " COMMISSIONS", BCOM(1), BCOM(2), BCOM(3)
63 END

```

DIM statements are not needed here.

For each salesman, read his model A and B sales.

Sales data for each salesman.

Commission for model A sales.

Model B sales above ten cars.

Over ten cars the commission is \$145 per car.

The program contains one loop. In the loop we read the data and compute the commissions. In lines 27 to 36 the commissions for model B sales are determined in a special way to account for the stipulation that the commission rate increases from \$95 to \$145 for all sales above ten model B cars. We now execute the program

```

RUN
SUPER AGENCY SALES REPORT

MODEL A SALES ERIC RON JEFF
COMMISSIONS 750 1125 500
MODEL B SALES 8 5 12
COMMISSIONS 760 475 1240

```

This report uses the four zones to create the four columns of information. Descriptive information appears in the first zone. Sales data pertaining to each of the three salesmen appear in the remaining three zones.



Two serious limitations are evident from the sales report. First, if the agency had more than three salesmen, we would require more than the four available zones. We need a way of squeezing together the tabular display. Second, the earnings in dollars are not lined up the way financial information is usually displayed. The **PRINT TAB** and **PRINT USING** statements help, respectively, to alleviate these two shortcomings.

The **TAB** function can be used to bring the columns of the report closer together. The principle is the same as in setting tabs on a typewriter. The column selected for output is represented by the **argument** (in parentheses) of the **TAB** function. **TAB(K)** means move the cursor to the Kth print position. The argument K can be a constant or an expression between 0 and 255 inclusive. **TAB** positions greater than 63 are on succeeding lines. **TAB** may be used several times within a **PRINT** statement, and no punctuation is required after a **TAB** specification.

**REMEMBER:** No space is allowed between **TAB** and the parenthesis; **TAB (K)** is not permissible. The proper form is **TAB(K)**.

#### EXAMPLES

```
PRINT "COL 1";TAB(10)"COL 10"
COL 1 COL10
PRINT TAB(3)3TAB(8)8
 3 8
PRINT TAB(3.5)3;TAB(8.6)8
 3 8
```

```
PRINT TAB(-3)3
?FC ERROR
PRINT TAB(8)8;TAB(3)3
 8 3
PRINT TAB(8)8;TAB(11)3
 8 3
```

```
10 FOR K=1 TO 5
20 PRINT TAB(K); "TOWER"
30 NEXT K
```

```
RUN
TOWER
TOWER
TOWER
TOWER
TOWER
```

#### COMMENTS

COL 1 starts in the first print position; COL 10 in the tenth.  
No punctuation is required.  
The 8 is in the eighth print position.  
The expression in (?), the argument, is truncated; **TAB** uses an integer argument.

The argument cannot be negative.

Unexpected.

As expected.

The index K of the FOR-NEXT loop is the argument of the **TAB** function.



#### Example: Graphing an Equation

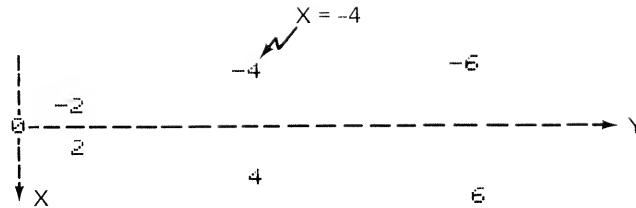
We use the **TAB** function to graph X versus Y for the parabola  $Y = X^2$ .

```
10 FOR X=-6 TO 6 STEP 2
20 Y=X * X
30 PRINT TAB(Y);X
40 NEXT X
45 END
```

```
RUN
```

#### COMMENTS

Graph the points at X = -6, -4, -2, 0, 2, 4, 6.



Each point on the graph is identified by the value of  $X$  at the point. The  $X$  and  $Y$  axes were drawn in manually. Similarly, the points can manually be connected to trace out the parabola. Notice that the parabola is upside down. This is because the  $X$  and  $Y$  directions are interchanged: here  $X$  is vertical and  $Y$  is horizontal. A more sophisticated approach to graphing is presented in a later chapter.



### Example: Pascal's Triangle

The numbers in this triangle have the property that every number is the sum of the number above it and the number to the left of the one above it.

```

10 REM PASCAL'S TRIANGLE
15 FOR D=1 TO 7
20 A(D,D)=1
25 NEXT D
30 FOR ROW=1 TO 6
35 K=0
40 FOR COL=1 TO ROW
45 A(ROW+1,COL)=A(ROW,COL)+A(ROW,COL-1)
50 PRINT TAB(K);A(ROW,COL);
55 K=K+5
60 NEXT COL
65 PRINT:PRINT
70 NEXT ROW

```

The loop of lines 15 to 25 sets the diagonal elements of the triangle to 1. The nested loops calculate and display the triangle of numbers. Line 45 is the algorithm for computing a new number  $A(\text{ROW} + 1, \text{COL})$  based on the number above it,  $A(\text{ROW}, \text{COL})$ , and the number to the left of the one above it,  $A(\text{ROW}, \text{COL}-1)$ . The `TAB` function is used to display the triangle with the variable  $K$  providing the number of tabs for even spacing. We now execute the program.

```

RUN
1
1 1
1 2 1
1 3 3 1
1 4 6 4 1
1 5 10 10 5 1

```

To appreciate the usefulness of the `TAB` function run the program without the `TAB(K)` in line 50 and note what a difference the `TAB` makes in the appearance of the triangle.

Pascal's triangle has several other properties. The descending diagonals are the same as the columns. The numbers in each row correspond to powers of 11: the first row is  $11^0 = 1$ , the second is  $11^1 = 11$ , the third is  $11^2 = 121$ , and so on. Similarly, the sums of the numbers in each row are powers of 2. For example, the third row is  $1 + 2 + 1 = 4 = 2^2$ . Two additional properties of Pascal's triangle deal with more advanced mathematics. The sum of the numbers along the ascending diagonals form the **Fibonacci** sequence, and the numbers in each row correspond to the coefficients of the **binomial expansion**. Can you edit the program to verify any of these properties?

The **TAB(X)** function controls the cursor's position within the 64 print positions on a line. The **POS(X)** function has the value corresponding to the cursor's position. **POS(X)** is therefore an integer between 0 and 63 inclusive. The argument **X** is a **dummy argument** and may be any number or numeric expression. When the cursor is in the usual position beneath the **READY**, it is in the 0 position.

#### COMMENTS

```
READY
PRINT POS(1) @ The argument 1 is a dummy argu-
 @ ment.
```

One possible application of the **POS** function is to display a report consisting of several columns with a prescribed spacing between columns. We now redo the headings of the Super Agency sales report.

```
PRINT "MODEL" TAB(POS(1)+5) "ERIC" TAB(POS(1)+5) "RON"
 TAB(POS(1)+5) "JEFF" TAB(POS(1)+5) "BRIAN"
MODEL ERIC RON JEFF BRIAN
```

In this case we displayed the headings five spaces apart. Notice that we can now comfortably fit five columns across the screen. We no longer depend on the four zones of the screen to provide the layout.

The screen displays up to 16 lines with 64 characters per line. This gives a total of 64 times 16 or 1024 print positions. These print positions are labeled 0 through 1023 and can be individually accessed using the **PRINT @** statement.

```
PRINT @ 0, 1 displays a 1 at the top left corner of the screen
PRINT @ 64, 2 displays a 2 on the second line along the left margin of the screen
PRINT @ 64*8, 3 displays a 3 midway along the left margin of the screen
```

The **PRINT @** statement may be used to graph an equation. In order to plot on successive lines, the location specification must take into account the presence of 64 print positions per line. The following program graphs the equation  $Y = X * X$ . It uses the **PRINT @** statement and presents an alternative to the version described earlier in this chapter in which the **TAB** function was utilized.

#### COMMENTS

```
10 CLS: I=0 Initialize the line counter to zero.
20 FOR X=-6 TO 6 STEP 2
30 Y=X*X
```

|                      |                                   |
|----------------------|-----------------------------------|
| 40 PRINT @ 64*I+Y, X | Graph at row I and position Y.    |
| 50 I=I+1             | Increment the counter to print in |
| 60 NEXT X            | the next row.                     |

Type in the program and check that its output is identical to the graph produced by the program that uses the TAB function.

It is important to note that the @ symbol in the PRINT @ statement may not be typed with the SHIFT key depressed. The symbol will appear to be correct on the screen, but an error will occur.

The format of the output may be selected in another way. The statement

PRINT USING "string"; item list

allows you to specify the format with which the item list is to be displayed. The actual format is given by the "string". For example, suppose we have a list of variables A, B, C whose values are \$27.216, \$351.951, and \$5. We wish to display these variables in a neat column.

|                             |                                |
|-----------------------------|--------------------------------|
|                             | <i>COMMENTS</i>                |
| 10 A=27.216: B=351.951: C=5 | Specify A, B, and C.           |
| 20 IMAGE\$="\$\$\$\$\$.##"  | Specify the "string" for PRINT |
| 30 PRINT USING IMAGE\$: A   | USING.                         |
| 40 PRINT USING IMAGE\$: B   |                                |
| 50 PRINT USING IMAGE\$: C   |                                |
|                             |                                |
| RUN                         |                                |
| \$27.22                     | Rounded to the nearest penny.  |
| \$351.95                    | Fractional penny dropped.      |
| \$5.00                      | Floating \$ sign.              |

The decimal points are lined up nicely; this can also be done slightly differently:

|                                |                                 |
|--------------------------------|---------------------------------|
|                                | <i>COMMENTS</i>                 |
| 10 A=27.216: B=351.951: C=5    |                                 |
| 30 PRINT USING "\$\$###.##"; A | In this version, line 20 is not |
| 40 PRINT USING "\$\$###.##"; B | needed.                         |
| 50 PRINT USING "\$\$###.##"; C |                                 |

The string "\$\$###.##" represents the numeric field that is selected for the variables A, B, and C. Each # specifies the position of a digit. In this case, numbers with more than two digits beyond the decimal point are rounded. The digits 5 and above are rounded up, and the digits 4 and below are dropped. The \$\$ characters place a \$ sign in front of the number.

**REMEMBER:** The string "\$\$###.##" will round to the nearest penny in a PRINT USING statement.

The following example further illustrates use of the PRINT USING statement:

```
10 PRINT USING "IN #### MANHATTAN WAS BOUGHT FROM THE
 INDIANS FOR $$$$.##", 1626, 24

RUN
IN 1626 MANHATTAN WAS BOUGHT FROM THE INDIANS FOR $24.00
```

In addition to the \$ and # characters, there are several other symbols used in conjunction with numerical and string data. These are summarized in Table 6.1 and illustrated in the following examples:

|                                     | <i>COMMENTS</i>                                  |
|-------------------------------------|--------------------------------------------------|
| 10 REM FORMATTING OF NUMERICAL DATA |                                                  |
| 20 INPUT IMAGE\$, NUMBER            | The "string" in PRINT USING is variable IMAGE\$. |
| 30 PRINT USING IMAGE\$; NUMBER      | The semicolon is required.                       |
| 40 GOTO 20                          | Infinite loop for data entry.                    |

We now execute the program and try different entries for the variables IMAGE\$ and NUMBER. Since the program continuously loops back to line 20, we only have to enter RUN once. To escape the loop, press BREAK.

|                         | <i>COMMENTS</i>                                                                                             |
|-------------------------|-------------------------------------------------------------------------------------------------------------|
| RUN                     | Request execution.                                                                                          |
| ? ###. #, 123. 45       | Data are entered.                                                                                           |
| 123. 5                  | 123.45 is rounded. Each # corresponds to one digit.                                                         |
| ? ###. #, 1234. 5       | The % indicates that specified field (IMAGE\$) is too small. Largest number that fits into ###. # is 999.9. |
| %1234. 5                |                                                                                                             |
| ? "#, ###. ##", 1234. 5 |                                                                                                             |
| 1, 234. 50              | The comma is inserted and trailing zeros added.                                                             |
| ? +###. #, 12. 3        |                                                                                                             |
| +12. 3                  | Display a leading + sign if number is positive.                                                             |
| ? ##. #+, 12. 3         |                                                                                                             |
| 12. 3+                  | A trailing + sign is now displayed.                                                                         |

**TABLE 6.1 Image specifiers for PRINT USING**

| Symbol   | Usage                                                                                                   | Example      |
|----------|---------------------------------------------------------------------------------------------------------|--------------|
| #        | Display a digit.                                                                                        | ###          |
| .        | Specify location of decimal point.                                                                      | ###.##       |
| ,        | Indicate where comma is to appear.                                                                      | #,###.##     |
| +        | For positive numbers, display a leading or trailing + sign; for negative numbers a - sign is displayed. | +###<br>###+ |
| -        | Display a leading or trailing - sign regardless if the number is positive or negative.                  | -###<br>###- |
| \$\$     | Display a \$ sign immediately preceding the leftmost digit.                                             | \$\$###.##   |
| **       | Replace leading blanks with asterisks.                                                                  | **###.##     |
| **\$     | Display a \$ sign preceding the leftmost digit and replace leading blanks with asterisks.               | **\$###.##   |
| ↑↑↑↑     | Display number in scientific notation (on the line printer, ↑ is []).                                   | #.#####↑↑↑↑  |
| !        | Display the first character of a string.                                                                | !            |
| %%       | Display the first two characters of a string.                                                           | %%           |
| %spaces% | Display N characters of a string. N equals 2 plus number of spaces.                                     | % %          |

|                           |                                                                                                 |
|---------------------------|-------------------------------------------------------------------------------------------------|
| ? -##. #, 12. 3           | A leading - sign is displayed even though 12.3 is positive.                                     |
| -12. 3                    |                                                                                                 |
| ? ##. #-, -12. 3          | -12.3 is negative, so a trailing - is displayed.                                                |
| 12. 3-                    |                                                                                                 |
| ? ##. #-, 12. 3           | 12.3 is positive, so instead of a - sign a trailing blank is displayed.                         |
| 12. 3                     |                                                                                                 |
| ? **##, 12                | Two asterisks are displayed.                                                                    |
| **12                      |                                                                                                 |
| ? **###, 12               | Fill leading blanks with asterisks.                                                             |
| ****12                    | Two asterisks appear plus two more for two leading blanks.                                      |
| ? \$####. ##, 12. 56      | \$ sign appears in front of leading blanks.                                                     |
| \$ 12. 56                 |                                                                                                 |
| ? \$#####. ##, 12. 56     | \$ sign appears immediately to left of first digit.                                             |
| \$12. 56                  |                                                                                                 |
| ? **\$#####. ##, 12. 56   | Combination of above; \$ sign appears to left of first digit; remaining blanks filled with *'s. |
| ****\$12. 56              |                                                                                                 |
| ? #. ####[####], 12. 56   | Exponential notation with no significant digit to left of decimal.                              |
| 0. 1256E+02               |                                                                                                 |
| ? ###. ####[####], 12. 56 | Two significant digits to left of decimal point.                                                |
| 12. 5600E+00              |                                                                                                 |
| ? _                       | To escape, enter BREAK.                                                                         |

The PRINT USING statement is also useful in displaying character information.

#### COMMENTS

|                                       |                                                            |
|---------------------------------------|------------------------------------------------------------|
| 10 REM FORMATTING OF CHARACTER DATA   |                                                            |
| 20 INPUT IMAGE\$, T\$, F\$, Z\$       |                                                            |
| 30 PRINT USING IMAGE\$; T\$, F\$, Z\$ |                                                            |
| 40 GOTO 20                            | Infinite loop for data entry.                              |
|                                       |                                                            |
| RUN                                   |                                                            |
| ? !, JOHN, FITZGERALD, KENNEDY        | The ! displays the first character of each string.         |
| JFK                                   |                                                            |
| ? ! ! ! , LYNDON, BAINES, JOHNSON     | Place a blank between the first characters of each string. |
| L B J                                 | The first two characters of each string are displayed.     |
| ? %% CARDS, SINK, NONE                | A combination of the ! and % formats.                      |
| CASINO                                |                                                            |
| ? ! ! % % THOMAS, STEARNS, ELIOT      |                                                            |
| T S ELIOT                             |                                                            |
| ? _                                   |                                                            |
| BREAK IN 20                           | To escape from program, press BREAK.                       |
| READY                                 |                                                            |
| > _                                   |                                                            |



#### Example: Checkbook Balancing

This program is used to balance a checking account. The program makes use of the PRINT USING statement. First the initial balance is entered. Then the individual transactions are entered; the month and day are typed in separated by a comma. Subsequently, the transaction is entered. A positive transaction is a deposit and a negative transaction is a check

charged against the account. To escape from the entry routine, enter END,END for the month and day. Two numeric and two string subscripted variables are used. The string variables store the month and day of the transaction, M\$(I) and D\$(I), respectively. The transaction array is TRANS(I), and the account balance is BAL(I), where BAL(0) is the initial balance. Once all the transactions and their dates have been entered into their respective arrays and the balances computed, the report is generated in lines 75 to 85. The loop's upper limit is I-1 (line 75), since the Ith transaction is a dummy transaction, which is not included. It is used to escape from the input loop. Following the listing of

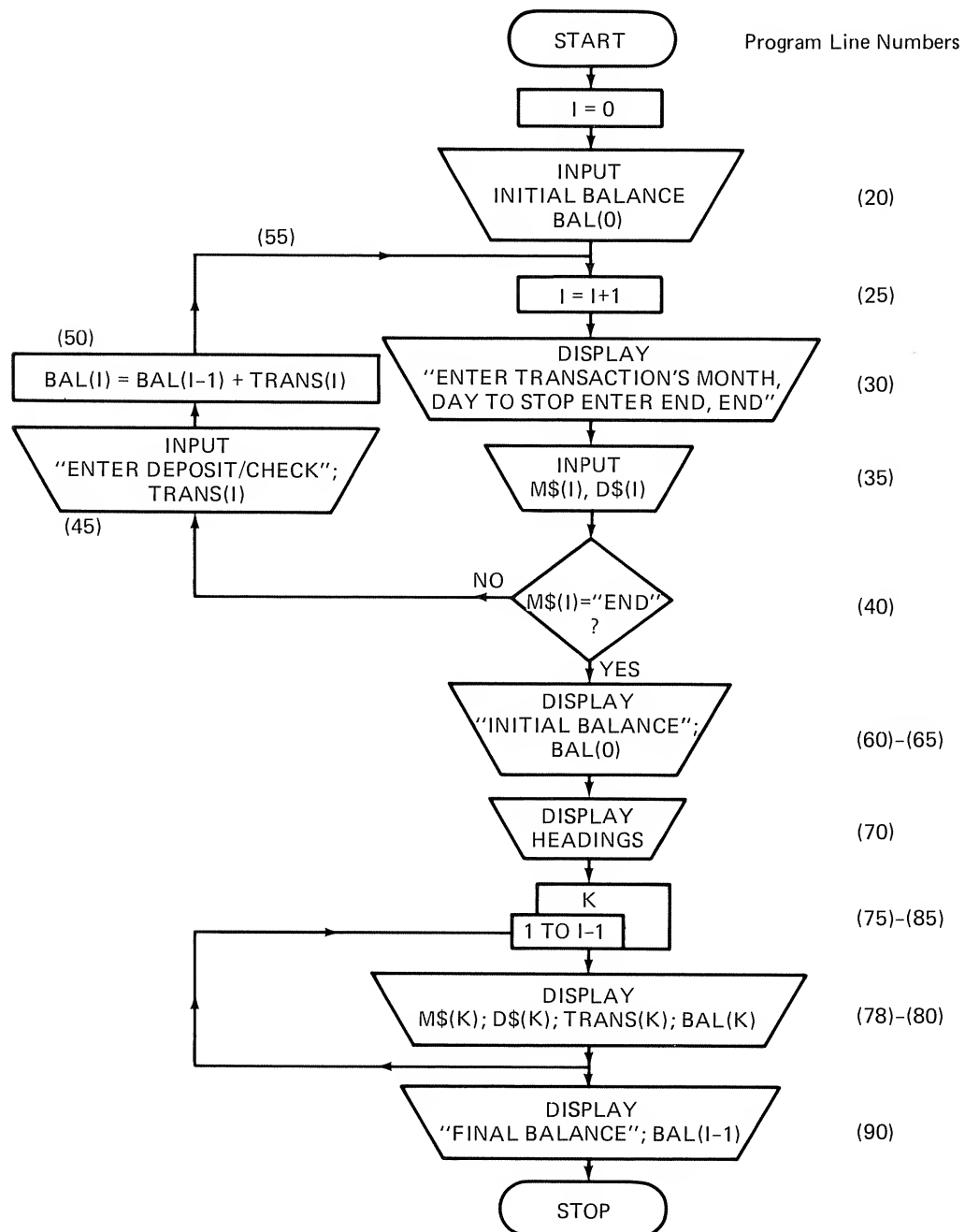


FIGURE 6.2 Checkbook balancing.

the program is a sample run, which illustrates the data entry and the resulting report. Type in the program and try it out on your own checkbook. You may wish to put a trace on the execution to gain full understanding of the logic. A flowchart of the program is shown in Figure 6.2.

```

10 DIM BAL(100), TRANS(100), D$(100), M$(100)
15 CLS: CLEAR 1000
20 INPUT "INITIAL BALANCE"; BAL(0)
25 I=I+1
30 PRINT "TRANSACTION MONTH, DAY FOR EX. MAY, 30; TO STOP ENTER END, END"
35 INPUT M$(I), D$(I)
40 IF M$(I)="END" THEN 60
45 INPUT "ENTER DEPOSIT(+)/CHECK(-)"; TRANS(I)
50 BAL(I)=BAL(I-1)+TRANS(I)
55 GOTO 25
60 CLS: PRINT "INITIAL BALANCE"; : P$="**$##, ####. ## DOLLARS"
65 PRINT USING P$; BAL(0)
70 PRINT: PRINT "DATE DEP/CHECK(-) BALANCE"
75 FOR K=1 TO I-1
78 PRINT USING "% %"; M$(K);
79 PRINT USING "%%"; D$(K);
80 PRINT USING "##, ####. ##-"; TRANS(K), BAL(K)
85 NEXT K
90 PRINT: PRINT "FINAL BALANCE $"; BAL(I-1)

```

RUN

```

INITIAL BALANCE? 1000
TRANSACTION MONTH, DAY FOR EX. MAY, 30; TO STOP ENTER END, END
? MARCH, 5
ENTER DEPOSIT(+)/CHECK(-)? -55
TRANSACTION MONTH, DAY FOR EX. MAY, 30; TO STOP ENTER END, END
? MARCH, 18
ENTER DEPOSIT(+)/CHECK(-)? 100
TRANSACTION MONTH, DAY FOR EX. MAY, 30; TO STOP ENTER END, END
? END, END

```

INITIAL BALANCE \*\*\*\*\*\$1,000.00 DOLLARS

| DATE  |    | DEP/CHECK(-) | BALANCE  |
|-------|----|--------------|----------|
| MARCH | 5  | 55.00-       | 945.00   |
| MARCH | 18 | 100.00       | 1,045.00 |

FINAL BALANCE \$ 1045

## 6.3 CASSETTE INPUT-OUTPUT

The TRS-80 supports two auxiliary memory devices, cassette tapes and mini disks. The major difference between a tape and a disk is that the tape is a sequential device, while a disk is a random-access device. In order to load into the computer a specific program stored on tape, it is necessary to forward the tape up to where the program of interest is stored. A program stored on disk may be accessed directly. Programs as well as data files can be stored on these auxiliary memory devices for later use. In this section we briefly discuss the input-output statements used in accessing a cassette tape.

To save a program on tape or to read a program from tape we use the **CSAVE** and **CLOAD** commands. These have been discussed earlier. The



PRINT #-1 and INPUT #-1 statements, respectively, print data on tape and read data from a cassette tape. The -1 specifies the device number, a cassette recorder. If two cassette recorders are used simultaneously, the device numbers are -1 and -2.

Storing data on tape is particularly useful when these data are to be updated occasionally or in data-processing applications involving lots of data. Small amounts of data that need little or no updating can conveniently be stored in a DATA statement within a program. The general form of the PRINT #-1 is

```
PRINT #-1, A, B, C, . . . , A$, B$, C$, . . .
```

The comma after the #-1 is required.

The list of items to be printed can contain numerical as well as string variables, but in total cannot exceed 255 characters. The general form of the INPUT #-1 statement is

```
INPUT #-1, A, B, C, . . . , A$, B$, C$, . . .
```

In order to read data from tape the variables in the PRINT #-1 and INPUT #-1 statements must be fully compatible. The INPUT #-1 statement must be identical to the PRINT #-1 statement that created the data file on tape.

The following sections of a program input from tape 100 names and year-to-date earnings, process the data, and then print on tape the updated earnings.

#### COMMENTS

```
50 INPUT "POSITION TAPE FOR READING. WHEN READY
ENTER GO";Q$
```

Press the PLAY button on recorder.  
This line is split to keep program  
and comments separate.

```
52 FOR K=1 TO 100
54 INPUT #-1, A$(K), E(K)
56 NEXT K
```

Read from tape 100 names and  
year-to-date earnings.

```
70 FOR K=1 TO 100
72 E(K)=E(K)+PAY(K)
74 NEXT K
```

Update year-to-date earnings; array  
PAY(K) must be entered in the  
lines between 56 and 70.

```
80 INPUT "REWIND TAPE. WHEN READY ENTER GO";Q$
82 IF Q$<>"GO" THEN 80
84 FOR K=1 TO 100
86 PRINT #-1, A$(K), E(K)
88 NEXT K
```

Press the RECORD and PLAY buttons on the recorder.

Print the names and the updated  
year-to-date earnings on tape.

Once the tape is positioned (line 50), the data are read in from the tape in lines 52 to 56. The earnings are updated in lines 70 to 74 by adding the current pay to the year-to-date earnings. The newly computed earnings are then stored on tape in lines 84 to 88. Line 80 stops execution to allow the operator to rewind the tape or to insert a new tape. It is usually good practice to save the previous data as a **back up**.

In the above example the number of records processed was 100. Generally, the number of records in a file is unknown. The last record of the file must be easily identifiable so that the INPUT #-1 can be properly terminated. A “dummy” record, also commonly referred to as a **flag**, is printed on tape at the end of the file. During the process of reading the tape into memory, each record is checked so that the last record can be identified. The last record may, for example, contain the employee’s name “LAST RECORD”. This name is then the last record’s identifying flag.

```

.
.
.
50 I=0
52 I=I+1
54 INPUT #-1, A$(I), E(I)
56 IF A$(I) <> "LAST RECORD" THEN 52
58 ...
.
.

```

Once the last record has been read from tape, execution proceeds in line 58. The total number of records processed is I, which includes the last (dummy) record.

## EXERCISES 9

- Find the error(s) in each of the following:

| Incorrect Instruction | Reason |
|-----------------------|--------|
| a. DATE 1, 2, 3       | _____  |
| b. DATA 1/2, 1/3, 1/4 | _____  |
| c. DATA, 3, 4         | _____  |
| d. DATA 3, 5 3, 6     | _____  |

- Combine the following two DATA statements into a single statement:

```

10 DATA 1, 2, 3
15 DATA 10, 20, 30

```

- Before executing the instructions, fill in the anticipated display and compare it with the actual display. Enter the instructions in the given sequence.

| Instruction      | Anticipated Display | Display |
|------------------|---------------------|---------|
| a. PRINT 1; 2    | _____               | _____   |
| b. PRINT 1, 2    | _____               | _____   |
| c. PRINT 1, , 2  | _____               | _____   |
| d. 10 PRINT 1, 2 | _____               | _____   |
| 20 PRINT, 3, 4   | _____               | _____   |
| RUN              | _____               | _____   |

| Instruction                                     | Anticipated Display | Display |
|-------------------------------------------------|---------------------|---------|
| e. PRINT TAB(1);6                               | _____               | _____   |
| f. PRINT TAB(1); 6                              | _____               | _____   |
| g. PRINT TAB(1), 6                              | _____               | _____   |
| h. CLS                                          | _____               | _____   |
| i. PRINT USING "###. ##"; 3. 1415               | _____               | _____   |
| j. PRINT USING "\$\$. ##"; 2222. 55             | _____               | _____   |
| k. PRINT USING "####. #"; 460                   | _____               | _____   |
| l. PRINT USING "!"; "TO BE OR NOT TO BE"        | _____               | _____   |
| m. PRINT USING "! !"; "PETER"; "JOSEPH"         | _____               | _____   |
| n. PRINT POS(1)                                 | _____               | _____   |
| o. PRINT POS(1), POS(2), POS(3), POS(4), POS(5) | _____               | _____   |
| p. PRINT TAB(10)POS(10)                         | _____               | _____   |
| q. PRINT TAB(10), POS(10)                       | _____               | _____   |
| r. PRINT @ 0, "HOME"                            | _____               | _____   |
| s. PRINT @ 1024/2+64/2, "MIDSCREEN"             | _____               | _____   |

4. What output will the following program produce?

```

10 DATA 2, 4, 6, 8
20 READ A, B
30 RESTORE
40 READ W, X, Y, Z
50 PRINT A; B; W; X; Y; Z

```

- Write a program to input a date as three variables, M, D, and Y. For example, 6/11/1940 will store M, D, and Y as 6, 11, and 1940, respectively. Output the date entered in the form JUNE 11, 1940. The names of the months are to be stored in a DATA statement.
- READ a list of numbers. Place the positive numbers of the DATA list into an array P and the negative numbers into the array N. Display the arrays P and N.
- Write a program to graph the equation  $Y = X^3 + X^2 - X + 2$  for  $X = 0, 0.5, 1, \dots, 3$ . Each point on the graph is to be identified by the value of X at the point.
- Write a program to display in tabular form the numbers 1 through 10, their squares, square roots, cubes, and cube roots. The columns are to be 12 spaces apart.
- Modify the car agency sales report program to include the total number of cars sold by each salesman and the total commission earned by each salesman.
- Modify the car agency sales report program to include a fourth salesman. Use the TAB function to lay out the report, taking advantage of the entire screen. Assume the fourth salesman sold no model A cars and 14 model B cars.
- What display does the following program produce?

```

10 FOR I=1 TO 10: PRINT TAB(I);"*":NEXT I

```

Can you explain it?

- Print a list of integers on a cassette. Copy this list from the cassette onto a second cassette with all duplicates removed.
- Write a program to create an **inventory file**. Each record includes the following items: part number, number on hand, price, vendor's name (20 characters). Once the file is created on tape, display it on the screen.

14. Write a program to create a **mailing list** on tape. Make provisions for initially entering names and addresses from the keyboard and subsequently adding and deleting names from the keyboard. The mailing list is to be displayed with an appropriate format and must include a dummy last record. Use your telephone book as a source of data.
15. Scan a **file of employees'** social security numbers, names, and birth dates. The company's mandatory retirement age is 65. Display a list of those employees who must retire within the next year, within two years, and within three years.
16. Write a computer program to build a file containing the earnings of the employees of a company. The company has 50 employees whose earnings are updated on a weekly basis from the keyboard. The file is to contain the name and year-to-date earnings of each employee. Each week the tape is read and an updated version is generated.

# chapter 7 | library functions

The BASIC language contains many different functions. Each function performs an operation that would otherwise take several statements in your program. Collectively, they form a library of functions. Like the arithmetic operations of addition or multiplication, the **library functions** make it possible to perform certain calculations that occur very frequently without having to program them separately each time. They are convenient to use and reduce the programming effort. Some library functions perform tasks that are quite complicated to program as they require advanced techniques. The functions are preprogrammed routines and are supplied with BASIC. Library functions are generally identified by a three- or four-letter name followed by an **argument** in parentheses. They are used as expressions in BASIC statements; properly applied they will save you many steps.

## 7.1 INT FUNCTION

The **INTEger** function, **INT(A)**, examines the argument **A** and returns a value equal to the greatest integer not larger than **A**. For example, the expression for the greatest integer less than or equal to 3.14 is **INT(3.14)**, which is 3. **INT(A)** can be referred to as “the greatest integer in **A**”. The argument **A** may be zero or any positive or negative number. The argument may also be an expression:

|                      | COMMENTS                                                                    |
|----------------------|-----------------------------------------------------------------------------|
| PRINT INT(3.14)      | 3 is largest integer not larger than 3.14.                                  |
| 3                    |                                                                             |
| PRINT INT(-3.14)     | -4 is largest integer not larger than -3.14. Note, -3 is larger than -3.14. |
| -4                   |                                                                             |
| R=6                  |                                                                             |
| PRINT INT(3.14*R(2)) | The argument may be an expression. INT (3.14*6*6) = 113.                    |
| 113                  |                                                                             |
| PRINT INT(X)         |                                                                             |
| 0                    | X is undefined; X = 0.                                                      |



### Example: Long Division

The **INT** function may be used to determine the quotient and remainder in long division. For example, dividing 16 by 5 gives a quotient of 3 and a remainder of 1. Converting 70 inches to feet and inches requires the division of 70 by 12, which yields 5 remainder 10, or 5 feet and 10 inches.

|                                  | <i>COMMENTS</i>                |
|----------------------------------|--------------------------------|
| 10 PRINT "ENTER A AND B FOR A/B" |                                |
| 20 INPUT A,B                     | Want to divide A by B.         |
| 30 Q=INT(A/B)                    | Compute the quotient.          |
| 40 R=A-B*Q                       | Compute the remainder.         |
| 50 PRINT "QUOTIENT="; Q          |                                |
| 60 PRINT "REMAINDER="; R         |                                |
|                                  |                                |
| RUN                              |                                |
| ENTER A AND B FOR A/B            | How many hours and minutes are |
| ? 355, 60                        | there in 355 minutes?          |
| QUOTIENT= 5                      | 5 hours, and                   |
| REMAINDER= 55                    | 55 minutes.                    |

How would you use this program to determine the number of days, hours, and minutes in 3000 minutes?



### *Example: Rounding to Any Desired Accuracy*

This program takes advantage of the INT function to round a number N up or down to as many digits D beyond the decimal point as is requested by the user.

```

10 INPUT "THE NUMBER YOU WISH TO ROUND"; N
20 INPUT "ACCURATE TO HOW MANY DIGITS BEYOND THE DECIMAL POINT"; D
30 IF N<0 THEN 50
40 PRINT INT(N*10^D+.5)/10^D: END
50 PRINT (1+INT(N*10^D+.5))/10^D: END

```

When this program is executed for several values of N and D, the following results are obtained

| Number N | Digits D | Result (Rounded N) |
|----------|----------|--------------------|
| 3.14     | 1        | 3.1                |
| 3.16     | 1        | 3.2                |
| -3.14    | 1        | -3.1               |
| 3.14     | 0        | 3                  |
| 314      | -1       | 310                |

Note, D = 1 rounds the first digit to the right of the decimal point. D = -1 rounds the first digit to the left of the decimal point.

In line 30 of the program we check if the number to be rounded is negative. Suppose N is 3.14 and D is 1. Let us **play computer** and check line 40.

|                           | <i>COMMENTS</i>                     |
|---------------------------|-------------------------------------|
| N=3.14: D=1               |                                     |
| PRINT N*10^D+.5           | $10^1 D = 10$ ; $N * 10^1 D = 3.14$ |
| 31.9                      | $N * 10^1 D + .5 = 31.9$            |
| PRINT INT(N*10^D+.5)      |                                     |
| 31                        | 31.9 is truncated.                  |
| PRINT INT(N*10^D+.5)/10^D |                                     |
| 3.1                       | The final answer.                   |

Can you play computer and check line 50 of the program for N = -3.14 and D = 1?

In a previous chapter we used the `PRINT USING` statement to round numbers. The “string” within the `PRINT USING` statement specified the desired number of digits to the right of the decimal point. The present program also rounds to the left of the decimal point. For example, in the last example of the above table of results we rounded 314 to 310.

**REMEMBER:** If  $X$  is an integer,  $\text{INT}(X)$  equals  $X$ . If  $X$  is a positive number,  $\text{INT}(X)$  equals the whole number part of  $X$ . If  $X$  is negative,  $\text{INT}(X)$  is the next lower whole negative number.

## 7.2 RND FUNCTION

The function **RND** generates random numbers between 0 and 1 and random integers greater than 0. The **RND** function is useful to simulate random events, for example, flipping a coin. The computer cannot toss a coin even once. But it can be programmed to simulate these tosses, that is, to produce outcomes that correspond to heads and tails. Since the outcome events of heads and tails are equally likely, the outcomes are random and the function **RND** can be used.

**RND(0)** generates a six-digit random number larger than 0 and less than 1. **RND(N)** generates a random positive integer between 1 and the integer portion of  $N$ .  $N$  must be a positive number less than 32768.

|                                        | COMMENTS                                                  |
|----------------------------------------|-----------------------------------------------------------|
| <code>PRINT RND(0)</code><br>.768709   | $N = 0$ . Random number between 0 and 1.                  |
| <code>PRINT RND(6)</code><br>4         | Throw a die and roll a 4; a random integer larger than 0. |
| <code>PRINT RND(52)</code><br>50       | Pick a card from a 52-card deck.                          |
| <code>PRINT RND(2)</code><br>1         | Flip a coin; for example, heads is 1 and tails is 2.      |
| <code>PRINT RND(0.5)</code><br>.938538 | The integer portion of the argument is used.              |
| <code>PRINT RND(6.75)</code><br>5      | Generates an integer between 1 and 6 inclusive.           |

When the statement **RANDOM** is executed in a program before the **RND** function is used, it will initialize the random-number generator to a new starting value. This will ensure that the **RND** function will produce a fresh sequence of random numbers that differs from any previous sequence.



### *Example: Generating Random Numbers Between Given Limits*

The following program generates any number of random numbers between the lower limit  $L$  and the upper limit  $U$ . The random number  $X$  is in the range  $L < X < U$ .

```

10 RANDOM
20 INPUT "DESIRED NO. OF RANDOM NOS. "; N
30 INPUT "LOWER, UPPER LIMITS"; L,U
40 FOR K=1 TO N
50 PRINT L+(U-L)*RND(0);
60 NEXT K

```

**COMMENTS**

Ensures randomness.

```

RUN
DESIRED NO. OF RANDOM NOS. ? 3
LOWER, UPPER LIMITS? 1,3
1.8971 1.22692 1.88466

```

The numbers are all between 1 and 3.

```

RUN
DESIRED NO. OF RANDOM NOS. ? 3
LOWER, UPPER LIMITS? 1,3
2.88418 2.33853 1.98631

```

These three numbers differ from the above; a random process.

Try this program for L, U of 0, 10 and 0, 0.5. What happens if you enter -10, 0 for L, U? or 10, 0?

**Example: Tossing Heads and Tails****COMMENTS**

```

10 RANDOM
15 H=0:T=0
20 INPUT "NUMBER OF TOSSES"; N
30 FOR K=1 TO N
40 IF RND(2)=2 THEN 60
50 T=T+1: GOTO 70
60 H=H+1
70 NEXT K
80 PRINT N; "TOSSES" ; T; "TAILS"; H; "HEADS"

```

Generate a 1 or a 2 randomly; a 1 is tails, a 2 is heads.

```

RUN
NUMBER OF TOSSES? 100
100 TOSSES 41 TAILS 59 HEADS

```

In this program we arbitrarily assign the occurrence of the random number 1 to tails and 2 to heads. As the number of tosses increases, we expect the number of heads and tails to be approximately equal. Type in the program and execute it for N = 10, 100, and 1000. If we repeat the run for N = 1000 a second time, will the number of heads be the same as the first time?

**REMEMBER:** RND(0) gives a random number larger than 0 and less than 1. RND(N) gives a random integer greater than 0 and less than or equal to N; for example, if N = 5, the random integer is greater than 0 and less than 6.

**Example: Random Graphic Display**

In this example we again generate a 1 or a 2 randomly, but instead of assigning them to tails and heads, respectively, we print a blank if a 1 occurs and print a star if a 2 occurs. This pattern of blanks and stars is set into a rectangle of width W and height H.



```

10 RANDOM: CLS
15 INPUT "WIDTH AND HEIGHT OF DISPLAY"; W,H
20 FOR I=1 TO H
25 FOR K=1 TO W
30 IF RND(2)=2 THEN 45
35 PRINT " ";
40 GOTO 50
45 PRINT "*";
50 NEXT K
55 PRINT
60 NEXT I
65 END

```

A sample execution follows:

```

RUN
WIDTH AND HEIGHT OF DISPLAY? 12,6
* *****
* ***** *
* ** * * *
* * *****
* *** ***
*** * ****

```

Enter the program and execute it for  $W = 50$  and  $H = 12$ . You can experiment with several different display characters, such as dashes, periods, or slashes, by editing lines 35 and 45. Another variation in the graphic display is possible by editing line 30

```
30 IF RND(5)=2 THEN 45
```

On the average only one out of five random numbers produced by this line are 2's. The graphic display will therefore consist primarily of blanks. Try it, and then edit lines 30, 35, and 45 together to produce some TRS-80 art. More sophisticated techniques for producing graphic displays are presented in a later chapter.

### 7.3 MORE FUNCTIONS

Table 7.1 lists a number of available library functions in addition to INT and RND. You may have not heard of some of them, depending on your mathematical background. The list is included for reference. In each case the argument A may be in single or double precision and may be an integer, constant, or an expression.



#### *Example: A Bar Graph of the ABS Function*

The **ABS(X)** function returns a positive value regardless of whether the argument X is positive or negative. Both **ABS(5)** and **ABS(-5)** equal 5. In this example we plot the equation  $Y = \text{ABS}(X)$  in the form of a **bar graph**, also called a **histogram**. We let X take on values between -18 and 18 inclusive. To plot a bar graph, it is necessary to display a character such as \* on the screen Y times for each value of X. So if  $Y = 5$ , five \*'s will represent the value of Y in the form of a bar.

TABLE 7.1 Additional library functions

| Function | Description                                                                                                                                                                                                                                           |
|----------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| ATN(A)   | Arctangent of the argument A. This is the angle in radians whose tangent is A. To convert radians to degrees, multiply radians by 57.2958.                                                                                                            |
| COS(A)   | Cosine of A. Argument A must be in radians. If A is in degrees, use $\text{COS}(A \cdot 0.0174533)$ .                                                                                                                                                 |
| SIN(A)   | Sine of A. Argument A must be in radians. If A is in degrees, use $\text{SIN}(A \cdot 0.0174533)$ .                                                                                                                                                   |
| TAN(A)   | Tangent of A. Argument A must be in radians. If A is in degrees, use $\text{TAN}(A \cdot 0.0174533)$ .                                                                                                                                                |
| EXP(A)   | Computes the exponential function $e^A$ . This is the inverse of the natural logarithm function $\text{LOG}(A)$ ; i.e., $A = \text{EXP}(\text{LOG}(A))$ .                                                                                             |
| LOG(A)   | Computes the natural logarithm of A. A cannot be negative. This is the inverse of the exponential function, $\text{EXP}(A)$ ; i.e., $A = \text{LOG}(\text{EXP}(A))$ . To compute the logarithm of A to the base B use $\text{LOG}(A)/\text{LOG}(B)$ . |
| SQR(A)   | Computes the square root of A. Argument A cannot be negative. This function is identical to $A \uparrow .5$ , but is faster.                                                                                                                          |
| ABS(A)   | Absolute value of A. If $A > 0$ , $\text{ABS}(A) = A$ . If $A = 0$ , $\text{ABS}(A) = 0$ . If $A < 0$ , $\text{ABS}(A) = -A$ .                                                                                                                        |
| CBDL(A)  | Converts the expression in the argument A into a value in double precision.                                                                                                                                                                           |
| CINT(A)  | Identical to $\text{INT}(A)$ except A must be in the range -32768 to +32767.                                                                                                                                                                          |
| CSNG(A)  | Converts the expression in the argument A into a value in single precision.                                                                                                                                                                           |
| FIX(A)   | Truncates all the digits to the right of the decimal point. For $A > 0$ , $\text{FIX}(A) = \text{INT}(A)$ . For $A < 0$ , $\text{FIX}(A) = \text{INT}(A) + 1$ .                                                                                       |
| SGN(A)   | Equals 1 if $A > 0$ ; equals 0 if $A = 0$ ; equals -1 if $A < 0$ .                                                                                                                                                                                    |
| TAB(A)   | $\text{PRINT TAB}(A)$ moves the cursor to position A on the line. If $A > 63$ , cursor moves to next line. Argument A must be between 0 and 255 inclusive.                                                                                            |

```

10 FOR X=-18 TO 18 STEP 6
20 Y=ABS(X)
30 PRINT USING "###"; X;
40 FOR I=1 TO Y
50 PRINT "*";
60 NEXT I
70 PRINT
80 NEXT X

```

```

RUN
-18*****
-12*****
-6*****
0*
6*****
12*****
18*****

```

The numbers displayed along the margin of the bar graph are the X values. The number of \*'s in each bar reflects the value Y, that is, the absolute value of X. The absolute value of -18 is 18, and consequently 18 \*'s are displayed at -18. Similarly, ABS(18) is 18, and 18 \*'s are displayed at X = 18. For X = 0, no \*'s should appear, since ABS(0) is 0. Do you know why a single \* appears at X = 0?



### Example: The Rule of 72 Verified

In an earlier chapter we introduced the rule of 72. The rule states that the number of years needed for a bank deposit to double in value is approximately equal to 72 divided by the annual interest rate. In this example we will check this rule of thumb and investigate its accuracy. The exact doubling time is the logarithm of 2 to the base (1 + R), where R is the interest rate expressed as a decimal fraction. We compare the exact and approximate doubling times for interest rates ranging from 2% to 20%.

```

10 REM CHECK THE RULE OF 72
15 CLS
20 PRINT "YEARS TO DOUBLE"
25 PRINT "INTEREST RATE", "EXACT FORMULA", "RULE OF 72", "DIFFERENCE"
30 FOR I=2 TO 20 STEP 2
35 REM CONVERT INTEREST RATE TO DECIMAL FORM
40 R=I/100
45 A=LOG(2)/LOG(1+R)
50 B=72/I
55 PRINT I, A, B, A-B
60 NEXT I

```

The LOG function is used in line 45 to compute the exact time to double. The logarithm of 2 to the base (1 + R) is the quotient of LOG(2) and LOG(1 + R). Execution of the program yields the following results.

| RUN           |                 |            |            |
|---------------|-----------------|------------|------------|
| INTEREST RATE | YEARS TO DOUBLE |            |            |
|               | EXACT FORMULA   | RULE OF 72 | DIFFERENCE |
| 2             | 35.0027         | 36         | -.997295   |
| 4             | 17.673          | 18         | -.326956   |
| 6             | 11.8957         | 12         | -.104336   |
| 8             | 9.00648         | 9          | 6.4745E-03 |
| 10            | 7.27254         | 7.2        | .0725422   |
| 12            | 6.11626         | 6          | .116263    |
| 14            | 5.29007         | 5.14286    | .147209    |
| 16            | 4.67018         | 4.5        | .170179    |
| 18            | 4.18784         | 4          | .187835    |
| 20            | 3.80178         | 3.6        | .201784    |

The rule of 72 seems to be quite accurate as demonstrated by the fourth column of the table. The difference between the exact number of years and the years predicted by the rule is small. The rule is most accurate for an 8% interest rate. For interest rates less than 8%, the difference is negative. This means that the rule of 72 is overestimating the number of years to double. For interest rates larger than 8%, it underestimates the number of years. All in all, it is a very good rule of thumb.

## EXERCISES 10

- Before executing the instructions, fill in the anticipated display and compare it with the actual display.

| Instruction          | Anticipated Display | Display |
|----------------------|---------------------|---------|
| a. PRINT INT(3)      | _____               | _____   |
| b. PRINT INT(3.01)   | _____               | _____   |
| c. PRINT INT(-3.01)  | _____               | _____   |
| d. PRINT INT(3*A)    | _____               | _____   |
| e. PRINT FIX (5.5)   | _____               | _____   |
| f. PRINT FIX (-5.5)  | _____               | _____   |
| g. PRINT SGN (-5)    | _____               | _____   |
| h. PRINT SGN (5)     | _____               | _____   |
| i. PRINT RND(0)      | _____               | _____   |
| j. PRINT RND(A)      | _____               | _____   |
| k. PRINT RND(2)      | _____               | _____   |
| l. PRINT RND(6)      | _____               | _____   |
| m. PRINT RND(52.5)   | _____               | _____   |
| n. PRINT RND(-6)     | _____               | _____   |
| o. PRINT INT(RND(0)) | _____               | _____   |
| p. PRINT ABS(-5)     | _____               | _____   |
| q. PRINT ABS(5)      | _____               | _____   |
| r. PRINT ABS(0)      | _____               | _____   |
| s. PRINT CINT(16/3)  | _____               | _____   |
| t. PRINT CINT(45678) | _____               | _____   |
| u. PRINT CDBL(1/3)   | _____               | _____   |
| v. PRINT SQR(4)      | _____               | _____   |
| w. PRINT SQR(-4)     | _____               | _____   |

- What do the following expressions represent?
  - $X - Y * \text{INT}(X/Y)$
  - $\text{SGN}(X) * \text{INT}(\text{ABS}(X))$
  - $\text{ABS}(X - \text{FIX}(X))$
  - ON SGN(X)+2 GOTO 10, 20, 30
- Write a program that performs the same function as INT(X). Input to the program is a noninteger number X, and the output is the integer portion of X. Do not use the library function INT in the program.
- Write single statements to generate the following:
  - A random integer X, where  $10 \leq X \leq 20$ .
  - A random number X, where  $1 < X < 2$ .
- Modify the program presented in this chapter to generate N random numbers between given limits. Input N, L, and U, and now generate only integers in the range L to U, where L and U are to be included in the range.
- Write a program to throw a die N times and display the frequency of occurrence of each of the six possible outcomes. Modify the program to monitor throwing two dice N times.
- Write a program to create your own random-number generator. Follow the procedure:
  - Input a starting number ST for example ST = 11.
  - Input a constant, BASE, for example BASE = 17.
  - Input a constant, MULT, for example MULT = 5.
  - Compute a new value for ST = ADD + MULT \* ST.
  - Compute a new value for ST = remainder left after ST (from step d) is divided by BASE. At this point ST is an integer between 0 and BASE-1. *Hint:* The remainder of X/Y is  $X - Y * \text{INT}(X/Y)$ .

f. Compute  $Z = ST/BASE$ .  $Z$  has a value between 0 and  $(BASE - 1)/BASE$ . If  $BASE$  is much larger than 1, then  $(BASE - 1)/BASE$  is close 1.  $Z$  is the desired random number between 0 and 1.

g. Repeat steps d to f for additional random numbers.

Test this random-number generator for randomness. Use it to throw a die  $N$  times. Does the six come out approximately one-sixth of the time? *Hint:* The random number in step f needs to be converted into a random number in the range 1 to 6.

8. Test the randomness of the statement `RND(20)` by generating  $N$  random numbers and displaying the following properties of the stored random numbers.

- The average of the random numbers.
- The number of values less or equal to 10.
- The number of odd integers.
- The number of times a value appears twice in succession.

Compare the results obtained for  $N = 100, 500$ , and  $1000$  with the expected results.

9. Give BASIC statements that simulate the following operations:

- Finding the total of a roll of three dice.
- Choosing 3 cards from a deck of 52 cards, replacing each card after it is drawn.
- Choosing 13 cards from a deck of 52 cards without replacing the cards after they are drawn.
- Spinning a roulette wheel.
- The birth dates of  $N$  people.

10. Write a program to play the game of Hi-Lo. The computer picks a number from 1 to 100 for you to guess. After each guess, the computer responds by telling you whether the guess was too high or too low. The computer keeps track of the number of guesses you take and informs you of the number of guesses it took you to guess the number.

11. Generate three random integer numbers each in the range from 1 to 8. Do not allow for the random numbers to be repeated; that is, the three numbers must all be different.

12. Toss a coin  $N$  times and determine the longest run of heads. Try your program for  $N = 10, 100, 500$ , and  $1000$ .

13. Given a three-digit whole number, display its reversal. The number 123 is then displayed as 321. *Hint:* Use the `INT` function to isolate the digits.

14. The number 153 has an interesting property. It equals the sum of the cubes of its digits; that is,  $153 = 1^3 + 5^3 + 3^3$ . There are only four three-digit numbers (including 153) with this property. Find these four numbers; they are in the range 100 to 500.

15. A market survey indicates that a manufacturer can sell 100,000 toys if the selling price is \$1. For every cent he lowers the price he can sell an additional 5000 toys. Write a program to compute the gross sales for each selling price from \$1 to 50¢. Display the results in the form of a graph of selling price versus sales. What price yields the maximum sales?

16. The sine of an angle may be obtained from the series

$$\sin X = X - \frac{X^3}{3!} + \frac{X^5}{5!} - \frac{X^7}{7!} + \frac{X^9}{9!} - \dots$$

where  $X$  is measured in radians. The  $!$  sign is called **factorial**. For example, five factorial ( $5!$ ) is computed as  $5 \times 4 \times 3 \times 2 \times 1$  and  $3! = 3 \times 2 \times 1$ . Write a program that uses the first five terms of the series to compute the sine of an angle

measured in radians. In tabular form, compare the sines computed from this series and from the library function SIN for angles of 10, 30, 60, 90, 120, 150, and 180 degrees. (One degree equals 0.0174533 radians.)

17. Modify the bar graph program presented in this chapter to display a bar graph of the sine function in the range 0 to 360 degrees.
18. Write a program to plot a graph of the sine function using the TAB function. Display the graph for angles between 0 and 360 degrees.
19. Write a program to compute  $e^X$  using the first six terms of the following series:

$$e^x = 1 + X + \frac{X^2}{2!} + \frac{X^3}{3!} + \dots$$

where 3! (three factorial) is  $3 \times 2 \times 1$ . Compare the results for  $X = 0, 0.5, 1, 1.5, \dots, 3$  with the EXP(X) function.

20. Write a program to verify the following trigonometric relation:

$$\text{SIN}(2*A) = 2*\text{SIN}(A)*\text{COS}(A)$$

for the angles  $A = 0, 10, 20, \dots, 90$  degrees. Display the expression on the left side of the relation in one column and the expression on the right side in a second column. Then visually check to see if the values are the same.

# chapter 8 | subroutines

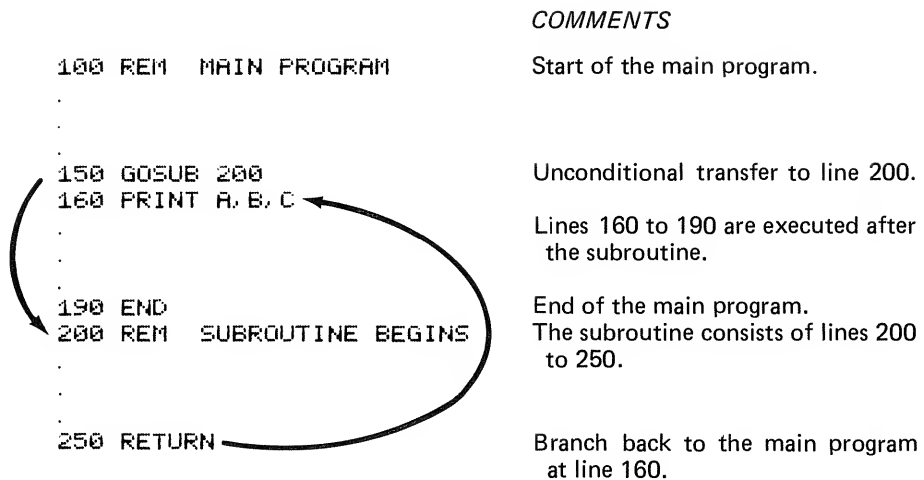
## 8.1 THE PURPOSE OF SUBROUTINES

A **subroutine** consists of a set of program statements that may be used repeatedly at different places throughout the program. Subroutines are usually written to carry out generalized procedures. For example, it may be necessary to sort arrays at different points within a program. Instead of repeating the statements required to perform a sort each and every time sorting is required within a program, it is possible to write the sort statements as a subroutine. This subroutine will then appear only once.

A subroutine can be an aid in writing shorter and more compact programs. The programmer can break his program into smaller logical components that are easier to work with, resulting in a more readable program. A subroutine can result in an economy of code for procedures that are to be performed repeatedly within a program or performed repeatedly in separate programs. Long complex programs that include no repetitive tasks may be broken into several segments. These segments then make up the complete program. They can be coded and debugged separately, and within the program one subroutine can refer to another subroutine.

## 8.2 UNCONDITIONAL TRANSFER TO SUBROUTINES

A subroutine is a sequence of statements within a program. Any line number within a program may be the start of a subroutine, as long as it is followed at some point by the statement **RETURN**. Access to the subroutine is gained from the **main** program. The statement **GOSUB n**, where **n** is the starting line number of the subroutine, causes the transfer to the subroutine. The computer then branches to the line indicated (line **n**) and executes all the statements up to the **RETURN** statement. Execution subsequently returns back to the line following the **GOSUB** statement. The **GOSUB** is an unconditional transfer statement. For example,



It may be necessary to have more than one subroutine. In that case we have a main program and two or more subroutines. Each subroutine is reached from the main program. If one subroutine calls on another subroutine, the subroutines are *nested*. For example, suppose the main program calls on subroutine A, which in turn calls on subroutine B. Once execution of B is completed, execution does not return directly to the main program, but rather to subroutine A. Specifically, execution returns to the statement following the `GOSUB` within subroutine A. Once the remaining lines of the subroutine A have been executed, execution returns to the main program at the line following the `GOSUB`. The main program terminates with an `END` statement; the subroutines each terminate with a `RETURN`. A subroutine may have more than one `RETURN` statement, and execution always returns to the statement following the `GOSUB` statement. We now type in a program that includes two nested subroutines.

```

100 PRINT "FROM MAIN PROGRAM"
110 GOSUB 200
120 PRINT "FINALLY, BACK IN MAIN PROGRAM AT LINE 120"
130 END
200 PRINT "ENTER SUBROUTINE A AT LINE 200"
210 GOSUB 300
220 PRINT "BACK IN SUBROUTINE A AT LINE 220"
230 PRINT "RETURNING TO MAIN PROGRAM"
240 RETURN
300 PRINT "ENTER SUBROUTINE B AT LINE 300"
310 PRINT "RETURNING TO SUBROUTINE A"
320 RETURN

```

This program demonstrates the execution of two nested subroutines. The main program consists of lines 100 to 130, subroutine A consists of lines 200 to 240, and subroutine B includes lines 300 to 320. It is a good idea to assign line numbers in different ranges (100's, 200's, and 300's) to individual subroutines. It makes it easier to follow the program. Run the program and observe the sequence in which the execution is performed.



```
RUN
FROM MAIN PROGRAM
ENTER SUBROUTINE A AT LINE 200
ENTER SUBROUTINE B AT LINE 300
RETURNING TO SUBROUTINE A
BACK IN SUBROUTINE A AT LINE 220
RETURNING TO MAIN PROGRAM
FINALLY, BACK IN MAIN PROGRAM AT LINE 120
```

**REMEMBER:** Every subroutine must have a RETURN statement.



#### *Example: Producing a Blinking Display*

On occasion it may be of interest to display a message that is sure to catch the user's attention. Examples of such instances include when an illegal entry is made, when a specific error occurs during execution, such as division by zero, or when one team is declared a winner in a computer game.

```
100 INPUT "TEXT TO BE DISPLAYED";A$
110 INPUT "DELAY TIME IN SECONDS";S
120 DELAY = 385*S
130 CLS
140 GOSUB 200
150 PRINT A$
160 GOSUB 200
170 GOTO 130
200 REM TIME DELAY SUBROUTINE
210 FOR I=1 TO DELAY: NEXT I
220 RETURN
```

Type in the program and then execute it. Enter your favorite phrase and a delay time of 2 seconds. The phrase will then blink every 2 seconds on the screen. The program is in an infinite loop as line 170 always transfers execution back to line 130, where the screen is cleared. To terminate the blinking display, press the BREAK key. During execution of the program, transfer is made to the time delay subroutine twice, once from line 140 and once from line 160. The first statement of the subroutine is in this case a REM statement. The subroutine consists of a FOR-NEXT loop. The number of times this loop is executed in line 210 depends on the desired duration of the delay between successive displays on the screen. Variable DELAY determines this duration. It is computed in line 120 as the product of 385 and the number of seconds S between successive displays. The number 385 is the number of times the computer executes the FOR-NEXT loop of line 210 in 1 second. You may use a timer to verify it.

This program clearly demonstrates the usefulness of subroutines. Transfer is made to the subroutine twice, but the subroutine appears only once and consists of lines 200 to 220. Each time the execution of the subroutine is completed, execution returns to the statement following the GOSUB. The computer "remembers" where it must return. Another important consideration is that this subroutine can be used in any other program in which a blinking message is to be displayed.

### 8.3 CONDITIONAL TRANSFER TO SUBROUTINES

The GOSUB is an unconditional transfer to a subroutine. It may be desirable to transfer conditionally to a subroutine.

|                                    | COMMENTS                                                                                                                                                                  |
|------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 10 GOSUB 200                       | Transfer unconditionally to the subroutine at line 200.                                                                                                                   |
| 10 IF A>B THEN GOSUB 200<br>20 ... | Conditional transfer: if A exceeds B, transfer to the subroutine at line 200; otherwise continue at line 20.                                                              |
| 10 ON E GOSUB 200, 300, 400        | Conditional transfer: if E equals 1 transfer to subroutine A at line 200; if E = 2, transfer to subroutine B at line 300; if E = 3, transfer to subroutine C at line 400. |



#### *Example: Computer-Assisted Instruction*

The following program is an arithmetic drill on short division. The program consists of a main program (lines 100 to 165) and two subroutines. The dialog between the computer and the student is established in the main program where the problems are posed. Depending on the student's response, the ON-GOSUB branches to the appropriate subroutine. If the student's answer is incorrect, execution transfers to the first subroutine (lines 200 to 250), where the computer states the correct solution and randomly picks one of three comments to encourage the student. Execution branches to the second subroutine, lines 300 to 330, if the answer is correct. The computer selects randomly and displays one of three preprogrammed compliments. This example demonstrates the use of two subroutines that are accessed by means of an ON-GOSUB statement. A flowchart of the program including line numbers is shown in Figure 8.1.

```
100 CLS: INPUT "HI. WHAT IS YOUR NAME"; N$
105 PRINT N$; " TODAY WE'LL PRACTICE SHORT DIVISION. "
106 PRINT "FOR EXAMPLE I'LL DISPLAY 126/21 AND YOU'LL TYPE IN"
107 PRINT "YOUR ANSWER 6 FOLLOWED BY 'ENTER'. YOU ONLY GET 1"
108 PRINT "CHANCE AT EACH OF THE 5 PROBLEMS! SO CONCENTRATE!"
110 K=0: COUNT=0: RANDOM
115 COUNT=COUNT+1: L=1
120 A=RND(20): C=A*RND(25)
125 INPUT "WHEN YOU ARE READY TYPE 'GO' AND PRESS ENTER"; G$: CLS
126 PRINT
130 PRINT "HERE'S PROBLEM NUMBER"; COUNT; ": "; C; "/" ; A; "=? "
135 INPUT "WHAT IS YOUR ANSWER"; B
140 IF C/A = B THEN L=2
145 ON L GOSUB 200, 300
150 IF COUNT<5 THEN 115
155 PRINT: PRINT N$; " YOU MISSED"; 5-K; "PROBLEMS OUT OF 5"
160 PRINT "IT WAS A PLEASURE WORKING WITH YOU --- SO LONG "; N$
165 END
200 REM SUBROUTINE: RESPOND TO INCORRECT ANSWER
205 ON RND(3) GOTO 210, 220, 230
210 PRINT "NOPE. YOU LOST YOUR COOL": GOTO 240
220 PRINT "YOU NEED TO CONCENTRATE MORE": GOTO 240
230 PRINT "COME ON "; N$; " YOU CAN DO BETTER"
```

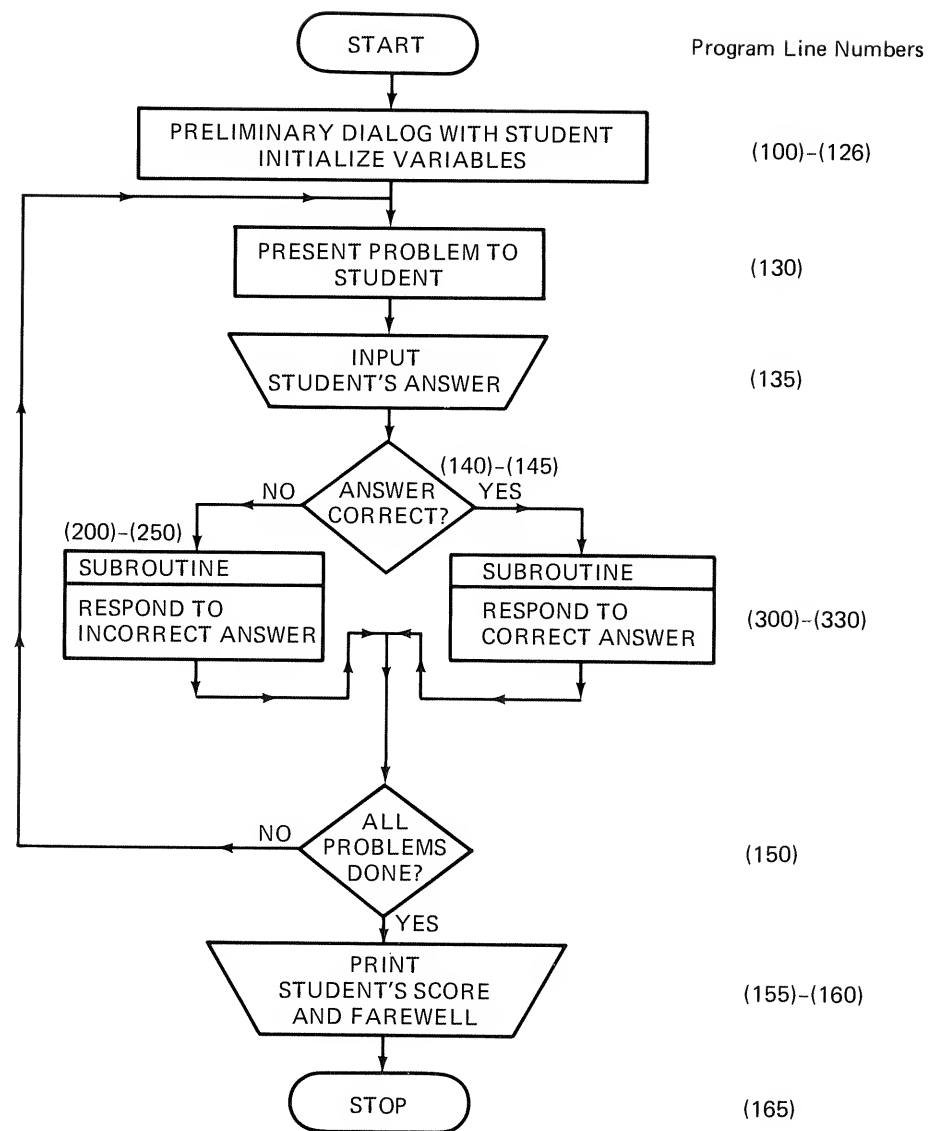


FIGURE 8.1 Computer-assisted instruction.

```

240 PRINT "THE CORRECT ANSWER IS"; C/A
250 RETURN
300 REM SUBROUTINE: RESPOND TO CORRECT ANSWER
304 K=K+1
307 ON RND(3) GOTO 310, 320, 330
310 PRINT "RIGHT ON --- YOU ARE DOING GREAT WORK!"; RETURN
320 PRINT "TERRIFIC! I LOVE IT!"; RETURN
330 PRINT N$; " YOU ARE A WHIZ KID"; RETURN

```

RUN

```

H1, WHAT IS YOUR NAME? MARION
MARION TODAY WE'LL PRACTICE SHORT DIVISION.
FOR EXAMPLE I'LL DISPLAY 126/21 AND YOU'LL TYPE IN
YOUR ANSWER 6 FOLLOWED BY 'ENTER'. YOU ONLY GET 1
CHANCE AT EACH OF THE 5 PROBLEMS! SO CONCENTRATE!
WHEN YOU ARE READY TYPE 'GO' AND PRESS ENTER? GO

```

```

HERE'S PROBLEM NUMBER 1 : 255/15=?
WHAT IS YOUR ANSWER? 15
YOU NEED TO CONCENTRATE MORE
THE CORRECT ANSWER IS 17
WHEN YOU ARE READY TYPE 'GO' AND PRESS ENTER? GO

HERE'S PROBLEM NUMBER 2 : 9/9=?
WHAT IS YOUR ANSWER? 1
MARION YOU ARE A WHIZ KID
WHEN YOU ARE READY TYPE 'GO' AND PRESS ENTER? GO

HERE'S PROBLEM NUMBER 3 : 180/15=?
WHAT IS YOUR ANSWER? 12
TERRIFIC! I LOVE IT!
WHEN YOU ARE READY TYPE 'GO' AND PRESS ENTER? GO

HERE'S PROBLEM NUMBER 4 : 136/17=?
WHAT IS YOUR ANSWER? 8
MARION YOU ARE A WHIZ KID
WHEN YOU ARE READY TYPE 'GO' AND PRESS ENTER? GO

HERE'S PROBLEM NUMBER 5 : 40/10=?
WHAT IS YOUR ANSWER? 4
RIGHT ON --- YOU ARE DOING GREAT WORK!

MARION YOU MISSED 1 PROBLEMS OUT OF 5
IT WAS A PLEASURE WORKING WITH YOU --- SO LONG MARION

```

In line 110 of the program, variables K and COUNT are initialized. K counts the number of correct answers given by the student, and variable COUNT keeps track of how many problems have been issued. The statement RANDOM reseeds the random-number generator and guarantees a new sequence of random numbers each time the program is executed.

In line 115 the counter is incremented and variable L is set to 1. L is a **flag**: L = 1 corresponds to an incorrect answer, and L = 2 corresponds to a correct answer given by the student.

Lines 120 to 135 generate the problem, display it, and accept the student's answer. The problem is created in such a way (line 120) that the answer is always an integer less or equal to 25.

Lines 140 to 145 check the student's answer and transfer execution to the appropriate subroutine. If the answer is incorrect, L = 1, execution transfers to the first subroutine at line 200. If the answer is correct, L = 2, execution is transferred to the second subroutine at line 300.

The first subroutine, lines 200 to 250, displays a message that is randomly picked among lines 210, 220, and 230. The correct answer is also given. The RETURN transfers execution back to the main program at line 150.

The second subroutine, lines 300 to 330, selects and displays randomly one of three compliments in response to the student's correct answer. Subsequently, execution returns to line 150 of the main program.

In line 150 of the main program, variable COUNT is checked; if it equals 5, all five problems have been answered and the final tally is presented in lines 155 to 160. Otherwise, execution returns to line 115, where the next problem is presented.

The END statement is usually optional. In this program it is however required. If the END in line 165 were omitted, execution would proceed from line 160 directly into the subroutine lines 200 to 250. That would make no sense, and in addition an RG ERROR

would occur, indicating a RETURN without GOSUB. In the absence of the END, execution of the subroutine is invoked without a GOSUB statement. A RETURN is then impossible to execute.

---

**REMEMBER:** Programs that include subroutines generally require an END statement.

## 8.4 A FINAL COMMENT

The examples we have studied only begin to illustrate the many ways subroutines can be used by the programmer. In many instances, a program may involve many calculations that are exceedingly involved and complex. For example, an analyst might be concerned about the financial future of a company. Many factors affect the company; among them are the cost of materials and manufacturing, inventory control, sales, and the share of the market. Using these and other characteristics, the analyst will attempt to forecast the company's future. The resulting total program can be very substantial. In such a case it is good practice to break the program down into separate segments. Each of these could then be written as a separate subroutine, and the main program would be a mere skeleton consisting of little more than a series of GOSUB statements. In many instances it is even convenient to perform input-output operations in separate subroutines. The advantages of such an approach include the following:

1. It is easier to write several short subroutines than one very long program.
2. Subroutines can be debugged and tested separately. Errors are located more readily in this manner.
3. Modifications to the program can be made more easily. It is possible to add a subroutine or to modify an existing one without affecting the rest of the program.
4. Subroutines make it more convenient to read a program or document its function.

## EXERCISES 11

1. Enter the following program. Before executing it, anticipate the output it will produce.

```
10 PRINT "START"
20 GOSUB 70
40 PRINT "BACK IN MAIN"
50 END
70 PRINT "SUBROUTINE 1"
75 RETURN
```

Anticipated display \_\_\_\_\_

Actual display \_\_\_\_\_

2. To the above program, add the line 25 GOSUB 70. What output will the program now produce? Delete line 25.

3. In the program of problem 1, delete line 50. What output will the program now produce? Reenter line 50.
4. Add the following statements to the program of problem 1.

```

30 GOSUB 80
80 PRINT "SUBROUTINE 2"
85 RETURN

```

Before executing the program, anticipate its output.

5. Add the following statement to the program of problem 4: 72 GOSUB 80. Anticipate the output of the new version of the program.
6. Add the statement 45 GOTO 10 to the program of problem 5. Now what happens?
7. Write a subroutine that examines an array of 10 numbers and outputs how many numbers are negative. Input the array in the main program.
8. Modify the program to produce a blinking display (presented in this chapter) to input the number of times the message is to be displayed.
9. Extend the blinking display program to operate like a **digital clock**. The display is to be the actual time.
10. Write a program to simulate a dice game. For each roll of the dice, use a subroutine that generates two random numbers between 1 and 6. On the first toss you win with a total of 7 and the computer wins with a total of 12. Any other sum becomes your point. You continue to throw, trying to match this point. If you roll a 7 in the process, the computer wins. Keep a tally of games won and lost.
11. Write a subroutine to compute and output the **mean** and **standard deviation** of data stored in an array A. Input the array in the main program. Use the following mathematical relationships for the computation.

$$\text{MEAN} = \frac{\text{SUM A}}{N}$$

$$\text{STANDARD DEVIATION} = \sqrt{\frac{\text{SUM A}^2 - (\text{SUM A})^2/N}{N - 1}}$$

where SUM A = sum of all data points in array A

SUM A<sup>2</sup> = sum of the squares of all the data points

N = number of data points; the number of elements of array A

Write a main program to input data and a subroutine to perform a **data validity check**. If the number is negative or greater than 100, the subroutine displays an appropriate message and returns to the main program for more data entry. The mean and standard deviation are to be computed and displayed in a second subroutine.

12. Write a subroutine for calculating **compound interest** using the formula

$$A = P \left( 1 + \frac{R}{N} \right)^{NT}$$

where A = accumulated amount (principal plus interest)

P = invested principal

R = annual interest rate (in decimal notation)

T = number of years

N = number of times the interest is compounded each year.

Values for these variables are to be entered in the main program. Use the following values:  $P = \$100$ ,  $R = 0.05$  (5%),  $T = 10$  years,  $N = 1, 2, 4, 8, 16, 32, 64, 128$ , and 365. Print a table of  $N$  and  $A$ .

13. Write a subroutine to compute **N factorial** ( $N!$ ). For example, 4 factorial ( $4!$ ) equals  $4 \times 3 \times 2 \times 1 = 24$ . Input  $N$  in the main program, and output  $N!$  from the subroutine.
14. Write a subroutine to calculate  $e^x$  using the formula

$$e^x = 1 + \frac{X}{1!} + \frac{X^2}{2!} + \frac{X^3}{3!} + \frac{X^4}{4!} + \frac{X^5}{5!} + \dots$$

In the main program, input  $X$  and the number of desired terms from the **infinite series**. Use the subroutine of the previous problem to compute the factorial terms. The program to compute  $e^x$  will therefore include **nested subroutines**.

15. Extend the program that computes  $e^x$  to include a **convergence criterion**. Instead of specifying the number of terms of the series that are to be included, have the subroutine include as many terms as are required until the relative difference between successive terms is no more than some prespecified value  $C$ . The relative difference is the absolute value of the ratio of the previous term minus the current term divided by the current term. Input  $C$  in the main program, and write a subroutine to determine whether the series has converged. The series has converged when the relative difference of successive terms is less than  $C$ . A typical value for  $C$  is 0.00001.

# chapter 9 | graphics

## 9.1 BACKGROUND

Graphic capabilities are quite important for educational, business, and recreational purposes. We can, for example, graph equations and bar charts or create computer art. The functions that make it all possible control the cursor and give the programmer the flexibility of printing characters on the screen at specific locations.

The cursor control instructions that have already been introduced are **CLS**, **CLEAR**, **TAB(X)**, **POS(X)**, and **PRINT @**. **CLS** clears the screen and moves the cursor along with the **READY** to the upper left corner of the screen. The **CLS** instruction can be part of the program. Upon its execution the screen is cleared and thus readied for subsequent output. The **CLEAR** key clears the display and returns it to a 64 character per line format. The **TAB(X)** function moves the cursor to the specified position **X** on the current line. If **X** is greater than 63, the cursor is moved to subsequent lines. The **TAB** function is useful in tabulating data or graphing equations. The **POS(X)** function returns a number from 0 to 63, indicating the current cursor position on the screen. The **POS** function can, for example, be used in conjunction with the **TAB** function to space data evenly across the screen. The **PRINT @** statement specifies exactly where on the screen the display is to appear. The specified location is a number between 0 and 1023. It corresponds to the 1024 ( $16 \times 64$ ) available print positions on the screen.

The **CLS**, **TAB**, **POS**, and **PRINT @** statements are most effective in formatting output, especially for reports. However, they are not quite as effective for plotting graphs or generating computer art. These instructions rely on a screen grid of 1024 print positions: 16 lines with 64 characters per line. This may be too crude and does not offer sufficient resolution. There are other graphic statements that offer a finer grid on the screen.

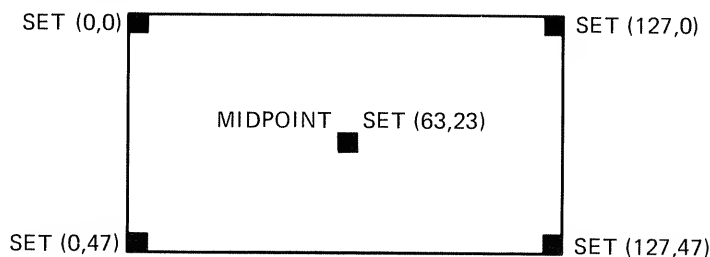
## 9.2 GRAPHING WITH SET(X,Y)

In graphing, we use a grid consisting of 128 points horizontally and 48 lines vertically. This grid of 6144 print positions offers the programmer a fine network of points. Each of these points can be addressed.

The **SET (X,Y)** function places a small rectangular block at position (X,Y) on the screen. **X** is the horizontal position 0 through 127 and **Y** the vertical position 0 through 47. **X** and **Y** can be numbers or expressions but



must be nonnegative. Location 0,0 is at the top left corner of the screen and 127,47 is at the bottom right corner.



SET(X,Y) may be used in the immediate mode and in the programming mode. Type in

```
SET (20,25)
```

The computer's response is a small rectangular point appearing at  $X = 20$  and  $Y = 25$ . Similarly, we can graph a point in the programming mode at  $X = 20$  and  $Y = 35$ :

```
10 SET(20,35)
RUN
```

This short program yields an additional point on the screen directly beneath the first point. The two points are separated by 10 vertical graph positions.

When graphing, it is usually desirable to remove all clutter from the screen including the READY. The following short program accomplishes this.

|               | <i>COMMENTS</i>                               |
|---------------|-----------------------------------------------|
| 10 CLS        | Clear the screen.                             |
| 20 SET(20,35) | One point is plotted at $X = 20$ , $Y = 35$ . |
| 30 GOTO 30    | Remain in an infinite loop.                   |

Line 30 represents a new **programming trick**. Upon execution of the program, the computer is hung up at line 30. The purpose of this maneuver is to suppress the READY from appearing on the screen at the end of the execution. We escape this hangup and obtain the READY by pressing the BREAK key. Graphing with the TAB and PRINT @ instructions permits use of any characters to denote points on the graph. The SET statement, in contrast, always plots with small rectangles. The fine grid accessible with SET instructions makes it possible to plot points so close to each other that they form lines. Compare, for example,

```
10 CLS
20 FOR X=0 TO 10: PRINT TAB(X)X : NEXT X
25 PRINT
30 FOR X=0 TO 30: PRINT TAB(X)"X" : NEXT X
40 FOR X=0 TO 60: SET(X,10) : NEXT X
50 GOTO 50
```

```
RUN
 0 1 2 3 4 5 6 7 8 9 10
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

---

To escape press the BREAK key. The first two lines of output were produced with the TAB function and the third using the SET statement. The SET displayed an actual line. To terminate execution, press BREAK.

**REMEMBER:** X and Y in SET(X,Y) cannot be negative.

X may be greater or equal to 0 and less or equal to 127.

Y may be greater or equal to 0 and less or equal to 47.

X = 0 is along the left edge of the screen.

Y = 0 is along the top of the screen.



### Example: Graph of Degrees Fahrenheit and Celsius

We wish to graph degrees Celsius versus degrees Fahrenheit. This graph will be a line that can be used to look up corresponding temperatures in the two temperature scales. The relationship to be plotted is

$$C = \frac{5}{9} (F - 32)$$

where C and F are temperatures in the Celsius and Fahrenheit scales, respectively.

#### COMMENTS

```
10 CLS
20 FOR F=32 TO 86
30 C=5*(F-32)/9
32 X=F
42 Y=C
50 SET(X,Y)
60 NEXT F
70 GOTO 70
```

Range of °F is 32 to 86, a total of 55 points on the graph.

The temperature conversion formula.

Without lines 32 and 42, line 50 would be SET(F,C).

Prevent the READY from appearing.

RUN

See Figure 9.1, version (a).

Execute the program and observe the shape of the graph. It is a line that starts at the top left corner of the screen at F = 32 degrees Fahrenheit and C = 0 degrees Celsius. The

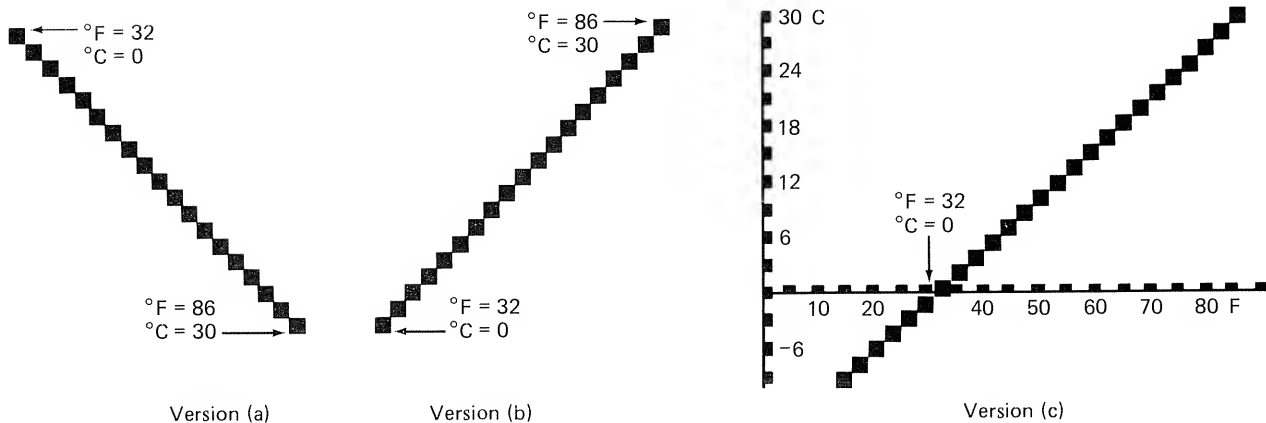


FIGURE 9.1 Several versions of the degrees Fahrenheit-Celsius graph.

line then progresses downward and to the right. The last point corresponds to  $F = 86$  degrees Fahrenheit and  $C = 30$  degrees Celsius. The graph is shown in Figure 9.1, version (a).

Normally, graphs are plotted with increasing values of  $Y$  in the upward direction. The screen of the TRS-80 is not laid out that way. The grid on the screen has  $Y$  increasing downward. We can overcome this by subtracting the computed value of  $C$  from 30. We insert line 40:

```
40 C=30-C
```

The value of  $C$  computed in line 30 is recomputed in line 40. For example, if  $C = 0$  in line 30, it will be recomputed in line 40 as  $C = 30$ .  $C = 30$  instead of  $C = 0$  is then graphed. Line 40 has the effect of starting the  $Y$ -axis at the bottom of the screen and pointing upward. Version (b) of Figure 9.1 shows an execution of the program with  $X = 32$  degrees Fahrenheit and  $Y = 0$  degrees Celsius now at the bottom left of the screen.

Finally, we modify the program to draw and label the axes. The axes are labeled with numbers in lines 11 to 14. Then the  $Y$ -axis is drawn in line 16 and dotted with markers in line 17. The  $X$ -axis is drawn in line 18, and the markers are placed along the  $X$ -axis in line 19. In line 20 we expand the range of the graph by including degrees Fahrenheit from 12 to 86. A flowchart of the program is shown in Figure 9.2.

```
10 CLS
11 PRINT @ 3, "30 C": PRINT @ 131, "24": PRINT @ 259, "18"
12 PRINT @ 387, "12": PRINT @ 515, "6"
13 PRINT @ 704, " 10 20 40 50 60 70 80 F"
14 PRINT @ 770, "-6"
```

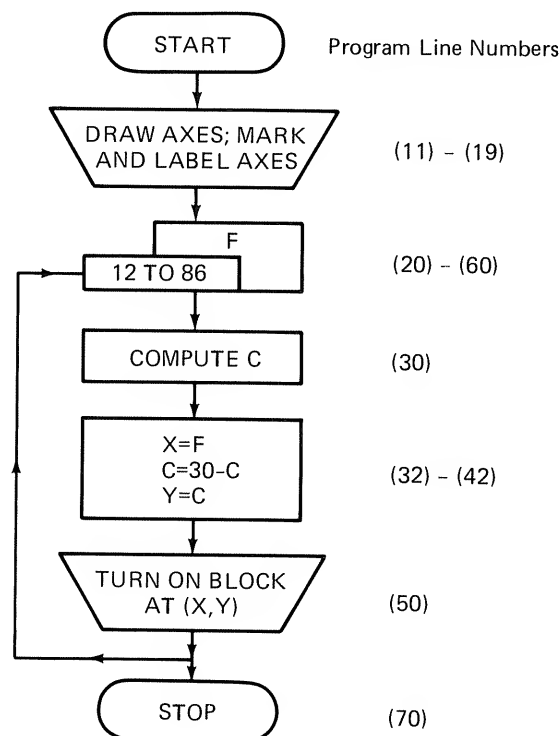


FIGURE 9.2 Conversion of degrees Fahrenheit to degrees Celsius—final version.

```

16 X=0: FOR Y=0 TO 40: SET(X,Y): NEXT Y
17 X=1: FOR Y=0 TO 40 STEP 3: SET(X,Y): NEXT Y
18 Y=30: FOR X=0 TO 86: SET(X,Y): NEXT X
19 Y=29: FOR X=0 TO 86 STEP 5: SET(X,Y): NEXT X
20 FOR F=12 TO 86
21 C=5*(F-32)/9
22 X=F
23 C=30-C
24 Y=C
25 SET(X,Y)
26 NEXT F
27 GOTO 70

```

Execution of this program yields version (c) in Figure 9.1. The markers are every five plot positions along the X-axis. The corresponding values of degrees Fahrenheit (F) and degrees Celsius (C) are displayed at every other marker. The scales differ for the two axes. There are three plot positions for each print position vertically. A multiple of three is therefore used. Horizontally, two plot positions correspond to one print position.



### Example: Bar Graph of a Frequency Count

As another example of the use of the SET function, we write a program to generate a bar graph of the frequency of occurrence of a 2, 3, 4 . . . 12 when two dice are thrown 100 times. There are a total of 36 possible ways that two dice can turn up the 11 different outcomes of 2, 3, 4 . . . 12 points. We expect, for example, one thirty-sixth of all the throws to yield 2 points between the two dice. A flowchart of the program is shown in Figure 9.3.

|                                        |                                                                 |
|----------------------------------------|-----------------------------------------------------------------|
| 10 CLS: RANDOM                         |                                                                 |
| 20 FOR K=1 TO 100                      | Throw the dice 100 times.                                       |
| 25 I=RND(6)+RND(6)                     | One random number for each die.                                 |
| 30 A(I-2)=A(I-2)+1                     | There are 11 outcomes numbered 0 to 10. Sum up the frequencies. |
| 40 NEXT K                              |                                                                 |
| 50 PRINT @ 3, "24": PRINT @ 131, "18"  | Print numbers along vertical axis:                              |
| 52 PRINT @ 259, "12": PRINT @ 387, "6" | frequency of occurrence.                                        |
| 60 PRINT @ 583, " 2 3 4 5 6"           | Label the X-axis.                                               |
| 65 PRINT " 7 8 9 0 1 2"                |                                                                 |
| 70 X=0: FOR Y=0 TO 24: SET(X,Y)        | Draw the Y-axis.                                                |
| 71 NEXT Y                              |                                                                 |
| 72 X=1: FOR Y=0 TO 24 STEP 6: SET(X,Y) | Place markers on the Y-axis.                                    |
| 73 NEXT Y                              |                                                                 |
| 76 Y=24: FOR X=0 TO 120: SET(X,Y)      | Draw the X-axis.                                                |
| 77 NEXT X                              |                                                                 |
| 80 FOR K=0 TO 10                       |                                                                 |
| 85 X=20+8*K                            |                                                                 |
| 90 FOR Y=24-A(K) TO 24: SET(X,Y)       | Draw the bar chart for each of the                              |
| 91 NEXT Y                              | 11 outcomes.                                                    |
| 95 NEXT K                              |                                                                 |

Lines 80 to 95 draw the vertical bars of the bar chart. Line 80 sets up a loop to be executed 11 times, once per outcome. The plot position of the bar along the X-axis is computed in line 85. For the first outcome ( $K = 0$ , throwing a 2),  $X$  is 20. For each subsequent outcome,  $X$  increases by eight plot positions; that is, successive bars are eight plot positions apart. The height of each bar is the frequency of occurrence,  $A(K)$ , of the  $K$ th outcome. The bars are graphed upward from the X-axis and not down from the top

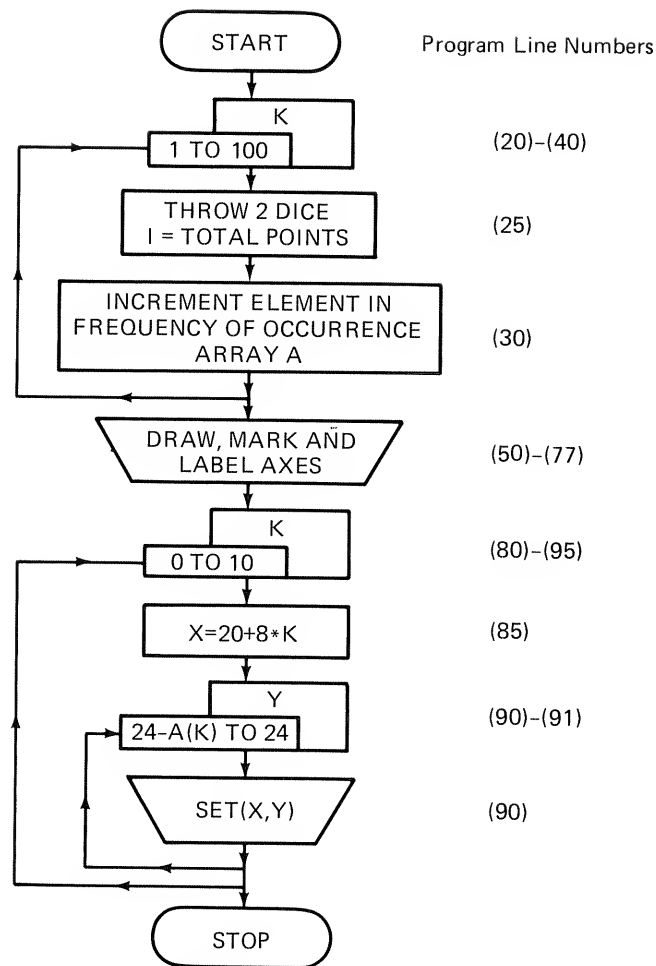
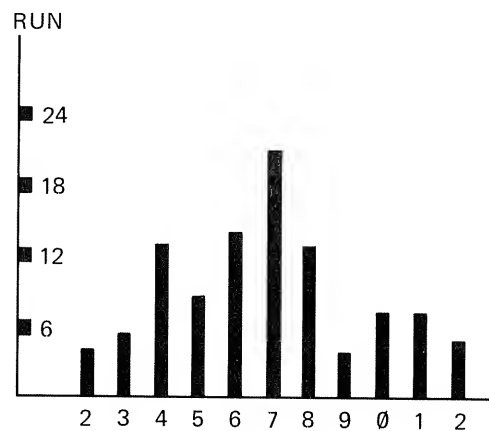


FIGURE 9.3 Bar graph of a frequency count.

of the screen. Therefore, the bar corresponding to a frequency of occurrence  $A(K)$  is drawn from  $Y = 24 - A(K)$  down to  $Y = 24$  (onto the X-axis).

We now execute the program.



The bar graph verifies our expectations: the seven is tossed most frequently. In this example of 100 tosses of two dice, the seven occurred 21 times. The least frequent tosses

are the two and the twelve, each of which occurred 4 times. Note, the outcomes of tossing 10, 11, and 12 are labeled as 0, 1, and 2 respectively along the X-axis of the bar graph. Do you expect the same frequencies to occur if we rerun the program and simulate 100 more tosses?

### 9.3 OTHER GRAPHICS FUNCTIONS

The **RESET(X,Y)** and **POINT(X,Y)** are two more functions that enhance the graphics capabilities of the TRS-80. The **RESET** function turns off a graphics block at the location specified by the coordinates **X** and **Y**. As with the **SET** function, **X** and **Y** must be nonnegative and be constants or expressions. In general, the integer portion of **X** and **Y** specifies the coordinates to be graphed.

|                             | <i>COMMENTS</i>       |
|-----------------------------|-----------------------|
| <code>SET(100, 40)</code>   | Graph a single point. |
| <code>RESET(100, 40)</code> | Erase that point.     |

The **POINT(X,Y)** function examines the graph location (X,Y) and checks whether it has been **SET** before. If the location (X,Y) has been **SET** (is on), **POINT(X,Y)** takes on the value -1. If the location (X,Y) has not been **SET** before, that is, if the location is off, the **POINT(X,Y)** function returns a 0.

|                                   | <i>COMMENTS</i>                      |
|-----------------------------------|--------------------------------------|
| <code>SET(100, 40)</code>         | Graph the point.                     |
| <code>PRINT POINT(100, 40)</code> | Check if (100,40) has been SET.      |
| <code>-1</code>                   | Yes, it has been SET.                |
| <code>RESET(100, 40)</code>       | Erase the point (100,40).            |
| <code>PRINT POINT(100, 40)</code> | The point (100,40) is no longer SET. |
| <code>0</code>                    |                                      |



#### *Example: Random Walk*

We visualize a mouse moving along in an open area. The mouse makes steps of the same length with equal probability in the forward and backward directions as well as to either side. The question is, what is the mouse's path if it never steps twice on the same location? This restriction may cause the mouse to be trapped when it finds itself surrounded by locations previously occupied. We simulate the random walk on the screen and start the mouse off at **X = 64**, **Y = 20**. When it reaches the edge of the screen (**X = 0**, or **X = 127**, or **Y = 0**, or **Y = 47**), the walk ends successfully. A flowchart of the program is shown in Figure 9.4. Corresponding line numbers of the program are shown in the flowchart.

```

10 ON ERROR GOTO 80
11 CLS: RANDOM
12 X=64: Y=20: K=0
15 SET(X,Y)

```

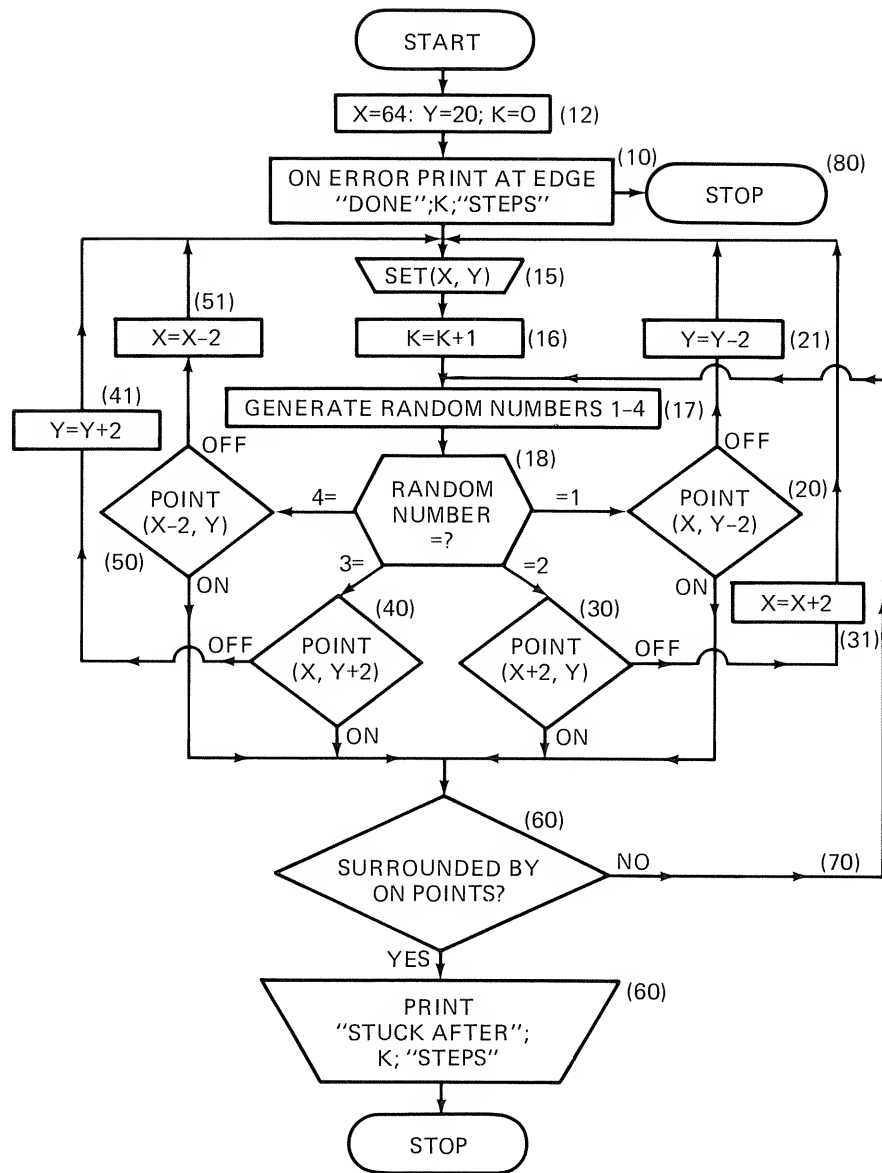
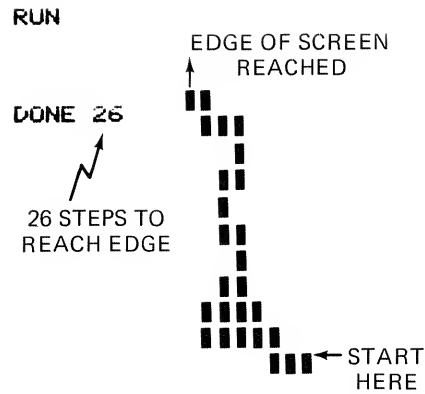


FIGURE 9.4 Random walk.

```

16 K=K+1
17 Z=RND(4)
18 ON Z GOTO 20, 30, 40, 50
20 IF POINT(X, Y-2) THEN 60
21 Y=Y-2: GOTO 15
30 IF POINT(X+2, Y) THEN 60
31 X=X+2: GOTO 15
40 IF POINT(X, Y+2) THEN 60
41 Y=Y+2: GOTO 15
50 IF POINT(X-2, Y) THEN 60
51 X=X-2: GOTO 15
60 IF POINT(X+2, Y) AND POINT(X-2, Y) AND POINT(X, Y+2) AND
 POINT(X, Y-2) THEN PRINT @ 0, "STUCK": K: GOTO 60
70 GOTO 17
80 PRINT @ 0, "DONE": K: GOTO 80

```



The random walk starts out at  $X = 64$  and  $Y = 20$  (lines 12 to 15). We then generate random numbers in the range 1 through 4 (line 17). Random numbers 1, 2, 3, and 4 correspond, respectively, to a forward (up) step, a step to the right, a backward (down) step, and a step to the left. Each step in the simulation corresponds to a jump of two blocks on the screen. In line 18, depending on the value of the random number  $Z$ , execution transfers to line 20, 30, 40, or 50. The `POINT(X,Y)` function is used to detect whether a location to which the mouse is about to move has previously been occupied. If that position is available, (`POINT(X,Y)=0`), then the step is taken by transferring to line 15. If the position is not available, a new random number needs to be generated in line 17. But before generating the new number it is necessary to check if the mouse is trapped. The `POINT` function is used in line 60 to check if there are any previously unoccupied locations either above, below, or to either side of the present position of the mouse. If the mouse is trapped, an appropriate message is displayed at the corner of the screen; otherwise, a new random number is generated in line 17 and the process of checking repeats.

The random walk is successfully completed when the mouse is at the screen's edge or attempts to go beyond. When the arguments  $X$  and  $Y$  in `SET` and `POINT` exceed their permissible limits, an error occurs. The occurrence of the error signals that the walk is complete. We take advantage of this feature and use the `ON ERROR GO TO` statement in line 10 to transfer to line 80 in the event of an error. Line 80 displays the message "DONE" along with the total number of steps taken to complete the walk.

Type in the program and observe the movements of the mouse. The motion can be slowed by inserting a delay prior to turning on the graphic block in line 15. Such a delay may consist of a dummy `FOR-NEXT` loop. Another possible variation is to change the jump size (presently at 2) to 1 or 3 or more. All lines containing the expressions  $X + 2$ ,  $X - 2$ ,  $Y + 2$ , or  $Y - 2$  would have to be modified accordingly. In the process of executing the program, some unique graphic patterns may develop.

## EXERCISES 12

- Before executing the following instructions, fill in what you anticipate the display will be. Add an explanation for any error you make.

| Instruction                     | Anticipated Display | Display |
|---------------------------------|---------------------|---------|
| a. <code>PRINT TAB(10)10</code> | _____               | _____   |
| b. <code>PRINT @ 10,10</code>   | _____               | _____   |
| c. <code>PRINT POS(10)</code>   | _____               | _____   |



| Instruction                     | Anticipated Display | Display |
|---------------------------------|---------------------|---------|
| d. PRINT TAB(10) POS(10)        | _____               | _____   |
| e. CLS                          | _____               | _____   |
| f. SET(10,10)                   | _____               | _____   |
| g. RESET(10,10)                 | _____               | _____   |
| h. 10 GOTO 10                   | _____               | _____   |
| RUN                             | _____               | _____   |
| i. BREAK                        | _____               | _____   |
| j. 10 SET(0,0)                  | _____               | _____   |
| 20 IF POINT(0,0) THEN PRINT"ON" | _____               | _____   |
| ELSE PRINT "OFF"                | _____               | _____   |
| RUN                             | _____               | _____   |
| k. 10 RESET(0,0)                | _____               | _____   |
| RUN                             | _____               | _____   |
| l. SET(10,5,10,6)               | _____               | _____   |
| m. RESET(10,10,1)               | _____               | _____   |
| n. SET(10,10)                   | _____               | _____   |
| o. POINT (10,10)                | _____               | _____   |
| p. PRINT POINT(10,10)           | _____               | _____   |

2. What displays do the following programs produce?

- a. 10 CLS: X=0  
20 FOR Y=0 TO 47: SET(X,Y): NEXT Y: GOTO 20
- b. 110 FOR X=1 TO 125  
120 FOR Y=1 TO 45  
130 SET(X,Y)  
140 NEXT Y,X  
150 Y=RND(23): X=RND(63)  
160 RESET(X,Y)  
170 RESET((63-X)+63,Y)  
180 RESET(X,(23-Y)+23)  
190 RESET((63-X)+63,(23-Y)+23)  
200 GOTO 150

3. a. Write a program to draw a vertical line at X = 63 and a horizontal line at Y = 23.  
b. Add markers every 5 units along the X - axis and every 3 units along the Y - axis.
4. Write a program to display a rectangle of width W and height H. The top left corner of the rectangle is to appear at X = A and Y = B. Input variables W, H, A, and B from the keyboard. How large may A, B, H, and W be?
5. Generalize the rectangle program to display N rectangles. Generate W, H, A, and B randomly. Input N from the keyboard. Select W between 1 and 10, H between 1 and 18, A between 0 and 117, and B between 0 and 29.
6. Write a program to plot a conversion graph from inches to centimeters (1 inch = 2.54 centimeters). Plot centimeters along the X-axis in the range 0 to 100 centimeters. Along the vertical axis, place markers every 6 inches. How many such markers are required?
7. Modify the random walk program of this chapter to eliminate the restriction that the same location cannot be occupied more than once. Run the program many times. Which edge of the screen does the mouse reach most frequently?
8. Write a program to simulate the following random walk: A drunk is staggering along an alley. He or she takes 2-foot-long steps with equal probability in the forward and backward direction. On the average, how far does the drunk move from the starting point at midscreen in N steps? Let N = 10, 50, 100, 500. Assume the drunk moves along the X-axis.

9. Write a program to generate 100 random numbers in the range 1 to 10. Output a bar chart showing the frequency of occurrence of each of the 10 random numbers. Do you expect the frequencies to be the same?
10. Write a program to plot two lines on the same graph. The equations to be graphed are  $Y = X$  and  $Y = 3X$ . Let  $X = 0, 1, 2, \dots, 10$ . Place markers and label the axes.

# chapter 10 | strings

## 10.1 REVIEW

The great power of the computer lies in its ability to not only manipulate numeric information but also process character information. Numeric variables are used to hold numeric quantities. These may be added, subtracted, multiplied, divided, evaluated in a function, or compared. Character information may consist of any combination of alphabetic and numeric characters, punctuation, and symbols. A string is a specific sequence of such characters. Strings do not enter into ordinary arithmetic operations. It certainly would make no sense to multiply the name of a person by ten or take its square root. However, we may want to process the name. In earlier chapters we have learned how to specify string variables, how to input and output them, how to compare them, and even how to “add” them. Addition of strings is called **concatenation** and refers to the process of appending one string to another. It is not an arithmetic addition. String variables differ from numeric variables in that they have a \$ symbol as the last character of the name. The following sequence of instructions reviews string manipulations.

```
A$="BIN"
PRINT A$
BIN
B$(1)="GO"
PRINT A$+B$(1)
BINGO
```

```
PRINT A$=B$(1)
0
PRINT A$<B$(1)
-1
```

```
10 INPUT "YOUR NAME"; N$
RUN
YOUR NAME? BOB BASIC
RUN
YOUR NAME? BASIC, BOB
?EXTRA IGNORED
PRINT N$
BASIC
```

### COMMENTS

Specify the string variable A\$.  
“BIN” is a string constant.  
Variable A\$ is displayed.  
String variables may be subscripted.  
The two string variables are concatenated.

The two string variables are compared. They are unequal.  
Alphabetically, “BIN” is before (less than) “GO”.

A one-line program to input your name.  
Request execution.  
Your entry.  
Request another execution.  
The string you enter now contains a comma; an error occurs; you need “ ” around entry.  
Only the part of the name prior to the comma is stored.

The last example indicates that a string constant must be enclosed in quotes when entered as INPUT if it contains a comma. Colons or leading blanks also require the quotes. The same rule regarding commas, colons, and leading blanks applies to INPUT # and READ-DATA statements.

## 10.2 ASCII CODES AND RELATED FUNCTIONS (ASC and CHR\$)

Two strings can be compared to determine which string variable or string constant is alphabetically closer to A. The characters are compared one at a time from left to right. Actually, the **ASCII code** of each character is compared. ASCII is short for American Standard Code for Information Interchange. It is a convention for identifying each character by a number. To perform operations on strings, this code is used within the programming language to designate the various alphanumeric characters. There are a total of 128 codes dealing with alphanumeric characters and an additional 127 codes (total of 255 codes) for the TAB function and for graphics.

A list of the ASCII codes and their usage is given in Table 10.1, which shows for each ASCII code the corresponding keyboard key and the corresponding character that it displays on the screen. For example, the code 65 corresponds to the letter A on the keyboard. Code 65 also displays the letter A on the screen. On the other hand, the ASCII code 1 corresponds to the BREAK key on the keyboard, while on the screen it produces no display. The most common codes may be summarized as follows:

| ASCII Code Number | Keyboard and Screen Display |
|-------------------|-----------------------------|
| 48-57             | Digits 0 to 9               |
| 65-90             | Letters A, B, . . . , Z     |
| 129-191           | Graphics symbols            |

The **ASC("string")** function returns the ASCII code of the first character of the "string." The "string" is the argument and must appear within the parentheses. It may be a string constant or a string variable. The **CHR\$(expression)** function returns a one-character string corresponding to the ASCII code specified by the expression. The argument must be numerical and must be a constant, a variable, or an expression in the range 0 to 255. The ASC and CHR\$ functions perform opposite operations and are illustrated in the following examples:

```
PRINT ASC("A")
65
PRINT ASC("AB")
65

PRINT ASC("0")
48
PRINT ASC(" ")
32
```

### COMMENTS

Display the ASCII code of letter A.  
ASCII code of letter A is 65.  
Display the ASCII code of "AB".  
Only ASCII code of first letter is displayed.  
Display ASCII code of digit 0.  
It is 48.  
Display ASCII code of a blank.  
It is 32.

TABLE 10.1 ASCII codes

| ASCII Code | Keyboard Key                | Screen Display                                           | ASCII Code | Keyboard Key        | Screen Display                                        |
|------------|-----------------------------|----------------------------------------------------------|------------|---------------------|-------------------------------------------------------|
| 0          | None                        | None                                                     | 58         | :                   | :                                                     |
| 1          | BREAK (also<br>SHIFT BREAK) | None                                                     | 59         | ;                   | ;                                                     |
| 2-7        | None                        | None                                                     | 60         | <                   | <                                                     |
| 8          | ←                           | Backspace & Erase                                        | 61         | =                   | =                                                     |
| 9          | →                           | None                                                     | 62         | >                   | >                                                     |
| 10         | ↓                           | Carriage Return                                          | 63         | ?                   | ?                                                     |
| 11         | None                        | Carriage Return                                          | 64         | @                   | @                                                     |
| 12         | None                        | Carriage Return                                          | 65         | A                   | A                                                     |
| 13         | ENTER (also<br>SHIFT ENTER) | Carriage Return                                          | 66         | B                   | B                                                     |
| 14         | None                        | Turn on cursor                                           | 67         | C                   | C                                                     |
| 15         | None                        | Turn off cursor                                          | 68         | D                   | D                                                     |
| 16-22      | None                        | None                                                     | 69         | E                   | E                                                     |
| 23         | None                        | Convert to<br>expanded print                             | 70         | F                   | F                                                     |
| 24         | SHIFT ←                     | Backspace cursor                                         | 71         | G                   | G                                                     |
| 25         | SHIFT →                     | Forward space<br>cursor                                  | 72         | H                   | H                                                     |
| 26         | SHIFT ↓                     | Linefeed ↓                                               | 73         | I                   | I                                                     |
| 27         | SHIFT ↑                     | Linefeed ↑                                               | 74         | J                   | J                                                     |
| 28         | None                        | Cursor to position<br>(0,0); convert to<br>regular print | 75         | K                   | K                                                     |
| 29         | None                        | Cursor to be-<br>ginning of line                         | 76         | L                   | L                                                     |
| 30         | None                        | Erase to end of<br>line                                  | 77         | M                   | M                                                     |
| 31         | CLEAR (also<br>SHIFT CLEAR) | Clear to end of<br>frame                                 | 78         | N                   | N                                                     |
| 32         | Space Bar                   | Space                                                    | 79         | O                   | O                                                     |
| 33         | !                           | !                                                        | 80         | P                   | P                                                     |
| 34         | "                           | "                                                        | 81         | Q                   | Q                                                     |
| 35         | #                           | #                                                        | 82         | R                   | R                                                     |
| 36         | \$                          | \$                                                       | 83         | S                   | S                                                     |
| 37         | %                           | %                                                        | 84         | T                   | T                                                     |
| 38         | &                           | &                                                        | 85         | U                   | U                                                     |
| 39         | '                           | '                                                        | 86         | V                   | V                                                     |
| 40         | (                           | (                                                        | 87         | W                   | W                                                     |
| 41         | )                           | )                                                        | 88         | X                   | X                                                     |
| 42         | *                           | *                                                        | 89         | Y                   | Y                                                     |
| 43         | +                           | +                                                        | 90         | Z                   | Z                                                     |
| 44         | ,                           | ,                                                        | 91         | ↑                   | ↑ or [                                                |
| 45         | -                           | -                                                        | 92         | None                | ↓                                                     |
| 46         | .                           | .                                                        | 93         | None                | ←                                                     |
| 47         | /                           | /                                                        | 94         | None                | →                                                     |
| 48         | 0                           | 0                                                        | 95         | None                | _ (Underline)                                         |
| 49         | 1                           | 1                                                        | 96         | SHIFT @             | @ (Lower case if available)                           |
| 50         | 2                           | 2                                                        | 97-122     | SHIFT A-<br>SHIFT Z | Lower case A-Z (if available);<br>otherwise uppercase |
| 51         | 3                           | 3                                                        | 123        | None                | ↑                                                     |
| 52         | 4                           | 4                                                        | 124        | None                | ↓                                                     |
| 53         | 5                           | 5                                                        | 125        | None                | ←                                                     |
| 54         | 6                           | 6                                                        | 126        | None                | →                                                     |
| 55         | 7                           | 7                                                        | 127        | None                | _ (Underline)                                         |
| 56         | 8                           | 8                                                        | 128        | None                | Space                                                 |
| 57         | 9                           | 9                                                        | 129-191    | None                | Graphics blocks                                       |
|            |                             |                                                          | 192-255    | None                | TAB(X) for X = 0, 1, . . . , 63,<br>respectively      |

```

PRINT CHR$(65) Display the symbol corresponding
A to code 65. It is letter A.
PRINT CHR$(ASC("A")) CHR$ and ASC have opposite func-
A tions.
PRINT ASC(CHR$(65))
65

```

The following program displays and performs the function of all the ASCII codes. Even though a delay has been incorporated into the program, you may wish to stop and **freeze the display** at any time. To stop the execution (without a BREAK), enter **SHIFT @**. To continue from where you left off, press any key (except the SHIFT and BREAK keys). Compare Table 10.1 to the screen's display during execution.

#### COMMENTS

```

10 REM DISPLAY ASCII CODES AND FUNCTIONS
20 CLS
30 FOR X=0 TO 255: PRINT X; CHR$(X),
40 FOR I=1 TO 150: NEXT I
50 NEXT X

```

Print code number and its screen display.  
Dummy loop for slowdown.

The next program decodes a message using the CHR\$ function.

#### COMMENTS

```

10 READ X
20 DATA 65, 32, 78, 73, 66, 66, 76, 69, 32, 73, 83, 32, 72, 65, 76, 70, 32, 65, 32, 66, 89, 84, 69
30 PRINT CHR$(X);
40 GOTO 10

RUN
A NIBBLE IS HALF A BYTE
?OD ERROR IN 10

```

An infinite loop; eventually there will be no more data to read.  
Out-of-data error in line 10.



#### Example: Maintaining a Status Message on the Screen

When lengthy and time-consuming internal computations are taking place, it is often a good idea to maintain a status message on the screen. This message indicates that the system is busy and perhaps gives some idea how close to completion it is. In this example we maintain a steady display of a loop counter during loop processing.

```

10 CLS: PRINT CHR$(26)
20 FOR I=1 TO 100
30 PRINT CHR$(27); TAB(15) "NOW PROCESSING LOOP"; I; "OF 100"
40 REM LOOP PROCESSING BEGINS
50 FOR D=1 TO 200: NEXT D
99 NEXT I

```

Line 10 clears the screen and moves the cursor down from the top of the screen. In line 30, CHR\$(27) moves the cursor up to the output line and TAB(15) centers the message. Line 50 is a loop that delays the display. When this routine is used in conjunction with loop processing, the loop of line 50 would not be included. What happens if the CHR\$(27) is not included in line 30? How about if CHR\$(26) is omitted in line 10? Can you explain the resulting displays?

**REMEMBER:** The ASC and CHR\$ functions complement each other; ASC(CHR\$(65)) = 65 and similarly CHR\$(ASC("A")) = "A".

### 10.3 CHARACTER MANIPULATION FUNCTIONS (LEN, LEFT\$, RIGHT\$, MID\$)

In this section we introduce several functions that may be used to analyze strings. The **LEN**("string") function returns the number of characters included in a string. The argument can be a string constant, a string variable, or an expression. Since a string may be up to 255 characters in length, **LEN**("string") is a number between 0 and 255 inclusive. The **LEFT\$**("string",N) function returns the first N characters of the string. The string in the argument may be a string constant, variable, or expression, and N may be any numeric constant, variable, or expression between 0 and 255. The function **RIGHT\$**("string",N) operates like the function **LEFT\$**("string",N) but returns the last N characters of the string. The **MID\$**("string",M,N) function returns a portion of "string" starting with the Mth position and containing N characters. For example, **PRINT MID\$("ABCD",2,3)** displays the string BCD. It starts with the second character of the string ABCD and is three characters long.

```
A$ = "TRS-80"
PRINT LEN(A$)
6
PRINT LEFT$(A$,3)
TRS
PRINT RIGHT$(A$,2)
80
B$=LEFT$(A$,7)
PRINT B$
TRS-80

PRINT LEN(B$)
6
PRINT LEFT$(A$,LEN(A$))
TRS-80

PRINT MID$(A$,2,3)
RS-
PRINT MID$(A$,LEN(A$),1)
0
```

#### COMMENTS

Specify string A\$.  
There are six characters in A\$.

The first three characters in A\$ are displayed.  
The last two characters in A\$ are displayed.  
The 7 is acceptable even though it exceeds LEN(A\$).

Is B\$ = 6 or 7 characters in length?  
B\$ has six characters.  
LEN function may be an argument for another function.

Select three characters from A\$, starting with the second.  
Display the last character in A\$.

The **LEN** and **LEFT\$** functions are used in the following program to create a unique display.

```
10 A$="TRS-80"
20 FOR M=1 TO LEN(A$)
30 PRINT LEFT$(A$,M)
40 NEXT M

RUN
T
TR
TRS
TRS-
TRS-8
TRS-80
```

The number of characters displayed on each line of output is controlled by M. M increases from 1 to 6; the length of the string "TRS-80" is 6.

**REMEMBER:** The functions LEFT\$, MID\$, and RIGHT\$ have strings as arguments and are used to isolate specific characters within the strings.



#### Example: Number of Words in a Text

In this program we input a text and check to determine how many words it contains. The number of blanks in a text is a direct measure of the words. The number of words equals the number of blanks plus 1. The text can be no longer than 255 characters. The CLEAR 600 reserves 600 bytes for string storage.

```
10 CLEAR 600: CLS: COUNT=0
20 INPUT "YOUR TEXT "; A$
30 FOR K=1 TO LEN(A$)
40 IF MID$(A$,K,1) <> " " THEN 60
50 COUNT=COUNT+1
60 NEXT K
70 PRINT "NUMBER OF WORDS": COUNT+1

RUN
YOUR TEXT? TO BE OR NOT TO BE
NUMBER OF WORDS 6
```

The number of characters in the entire text, LEN(A\$), determines the range of the loop (lines 30 to 60). The text's characters are compared one at a time to the blank " " in line 40. If a character equals a blank, the counter COUNT is incremented by 1. Enter the program and try it on your favorite phrase. Does the program work for a text consisting of a single word?



#### Example: Palindromes

A palindrome is a word that reads the same forward or backward, for example, OTTO or MADAM. The following program determines whether a specific string is a palindrome.

```
10 CLEAR 1000
20 INPUT "YOUR TEXT"; A$
30 FOR K=LEN$(A$) TO 1 STEP -1
40 B$=B$+MID$(A$,K,1)
50 NEXT K
70 IF A$=B$ THEN PRINT "IT IS A PALINDROME": END
80 PRINT "IT IS NOT A PALINDROME"

RUN
YOUR TEXT? MADAM
IT IS A PALINDROME
```

The loop of lines 30 to 50 rearranges the text A\$ and forms string B\$. This new string is string A\$ backward. If A\$ equals B\$, the text is a palindrome, and otherwise it is not. What happens if the text we input in line 20 consists of a single character? Try it. Can you think of any palindromes?





### Example: Coding a Message

To code a message, we add a certain number to the ASCII value of each of the characters of the message. This added value is the code. It is entered as part of the INPUT and is called N.

```

10 CLEAR 1000
20 INPUT "YOUR CODE VALUE"; N
30 INPUT "YOUR MESSAGE"; A$
40 FOR K=1 TO LEN(A$)
50 REM ISOLATE THE KTH CHARACTER
55 B$=MID$(A$, K, 1)
60 REM ADD N TO THE ASCII CODE OF THE KTH CHARACTER
65 B=N+ASC(B$)
70 REM CONCATENATE THE CODED CHARACTER TO THE CODED MESSAGE
75 CODE$=CODE$+CHR$(B)
80 NEXT K
90 PRINT "THE CODED MESSAGE: "; CODE$

```

```

RUN
YOUR CODE VALUE? 1
THE MESSAGE? ABC
THE CODED MESSAGE: BCD

```

This program works fine as long as the sum of the ASCII codes of the characters and N does not exceed 255 in line 65. If N is very large or if characters with high ASCII code values are used, it may be necessary to test B. If B exceeds 255, then specify it as  $B = B - 255$ .

Can you write a similar program to accept a coded message along with its code value and output the decoded message?

## 10.4 OTHER STRING FUNCTIONS

The **STRING\$(N, "character")** returns a string consisting of N identical characters. N is a numeric constant, variable, or expression, and the character can be any alphanumeric character or an ASCII code number. The **STRING\$** function may be used effectively to prepare graphic displays and bar graphs. The following examples illustrate the use of this function:

|                                       | COMMENTS                                                          |
|---------------------------------------|-------------------------------------------------------------------|
| PRINT STRING\$(5, "A")<br>AAAAA       | N = 5; the character is an A.                                     |
| PRINT STRING\$(5, 65)<br>AAAAA        | N = 5; the character is the ASCII code 65, which is the letter A. |
| PRINT STRING\$(100, "A")<br>?OS ERROR | An out-of-string space error occurs; CLEAR 100 is needed.         |

The request for a string of 100 A's caused an error. When the computer is first turned on, a **CLEAR 50** is automatically executed. The **CLEAR n** command resets all numeric variables to zero and all string variables to null. In addition, this command reserves n bytes for string storage. A request of 100

A's exceeds 50, the limit set automatically when the computer is turned on. To avoid the OS error, we first execute **CLEAR 100**.

The **FRE("string")** function returns the amount of unused string storage space. The argument string is a dummy argument. Any string constant, variable, or expression can be used as a valid argument.

|                                                              | COMMENTS                                                                                                               |
|--------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------|
| <b>PRINT FRE("A")</b><br>50                                  | When the computer is first turned on, 50 bytes are reserved for strings.                                               |
| <b>A\$=STRING\$(30, "A")</b><br><b>PRINT FRE("B")</b><br>20  | A\$ is a string of 30 A'S.<br>Any string is acceptable as an argument of FRE.<br>20 bytes are now left (50 - 30 = 20). |
| <b>CLEAR 300</b><br><b>PRINT LEN(A\$)</b><br>0               | A\$ is set to null and 300 bytes are now reserved.<br>A\$ is now a string of zero length.                              |
| <b>A\$=STRING\$(80, "A")</b><br><b>PRINT FRE("C")</b><br>220 | A\$ is 80 characters in length.<br>220 bytes remain unused.                                                            |
| <b>A\$=STRING\$(300, "A")</b><br>?FC ERROR                   | Illegal-Function-Call error.<br>A string cannot exceed 255 characters.                                                 |

The **FRE("string")** function is not to be confused with the **MEM** function, which returns the unused bytes in memory.

The **STR\$(expression)** converts a numeric expression or constant to a string. For example, if **PI = 3.1415**, then **STR\$(PI)** equals the string "3.1415". The leading blank is to allow space for a negative sign. Since **STR\$(PI)** is a string constant, only string operations may be performed with it and no arithmetic operations.

The function **VAL("string")** returns the number represented by the characters of the argument string. For example, **VAL("123")** equals the numeric constant 123. The **VAL** and **STR\$** functions perform opposite operations.

|                                                  | COMMENTS                                                                                 |
|--------------------------------------------------|------------------------------------------------------------------------------------------|
| <b>A=123</b><br><b>PRINT STR\$(123)</b><br>123   | Convert 123 to the string "123".<br>123 is a string.                                     |
| <b>PRINT STR\$(123)+1</b><br>?TM ERROR           | Type-Mismatch error.<br>Cannot add 1 to a string.                                        |
| <b>PRINT VAL("123")+1</b><br>124                 | <b>VAL("123")</b> is the number 123.<br>123 + 1 = 124.                                   |
| <b>B=VAL(STR\$(A))</b><br><b>PRINT B=A</b><br>-1 | A and B are equal since <b>VAL</b> and <b>STR\$</b> perform inverse operations.          |
| <b>PRINT LEN(STR\$(-3))</b><br>2                 | The string "-3" consists of two characters.                                              |
| <b>PRINT LEN(STR\$(3))</b><br>2                  | The 3 is positive and therefore has a leading blank; the string then has two characters. |

**REMEMBER:** The **VAL** and **STR\$** functions complement each other; **VAL(STR\$(7))** equals the number 7, and, similarly, **STR\$(VAL("8"))** equals the string "8".



### Example: Underlining a Title

The following program underlines all the nonblank characters in a title. A flowchart is shown in Figure 10.1.

```

100 CLEAR 600
105 INPUT "WHAT IS YOUR TITLE"; T$
110 CLS: N=0
120 PRINT T$
130 FOR K=1 TO LEN(T$)
140 IF MID$(T$,K,1)<>" " THEN N=N+1: GOTO 160
150 GOSUB 200
160 NEXT K
170 REM LOOP COMPLETED
180 GOSUB 200
190 END
200 REM SUBROUTINE TO UNDERLINE
210 PRINT STRING$(N, "-");
220 PRINT " ";
230 N=0
240 RETURN

RUN
WHAT IS YOUR TITLE? COMPUTERS ARE FOR KIDS
COMPUTERS ARE FOR KIDS

```

After the title is entered the screen is cleared. The title is displayed and the underlining appears on a separate line single spaced below the title. Does the program work for a title consisting of a single character or a title consisting of several words each separated by several blanks?

The program consists of a main program (lines 100 to 190) and a subroutine (lines 200 to 240). In the main program we examine each character in the title. The counter N is incremented if the character is not a blank. Once a blank character is encountered in line 140, transfer is made to the subroutine, where N minus signs are printed to underline N nonblank characters in the title. The STRING\$ function is used for this purpose. Then a blank is printed to properly break the underlining of the title beneath the blank. Execution subsequently returns to the main program within the loop at line 160. Each time transfer is made to the subroutine the nonblank characters between successive blanks are underlined. Once the loop is completed, transfer is made for the last time to the subroutine in line 180, and the last characters of the title are underlined.



### Example: Binary-to-Decimal Conversion

In this example we input a **binary number** consisting of zeros and ones. The computer then converts the binary number to its decimal equivalent. For example, the binary number 1101 is converted as follows:  $1101 = 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = 8 + 4 + 0 + 1 = 13$ . So binary 1101 is equivalent to decimal 13.

```

10 INPUT "BINARY NUMBER"; B$
20 N=0: DEC=0
30 FOR K=LEN(B$) TO 1 STEP -1
40 N=N+1
50 D$=MID$(B$,K,1)

```

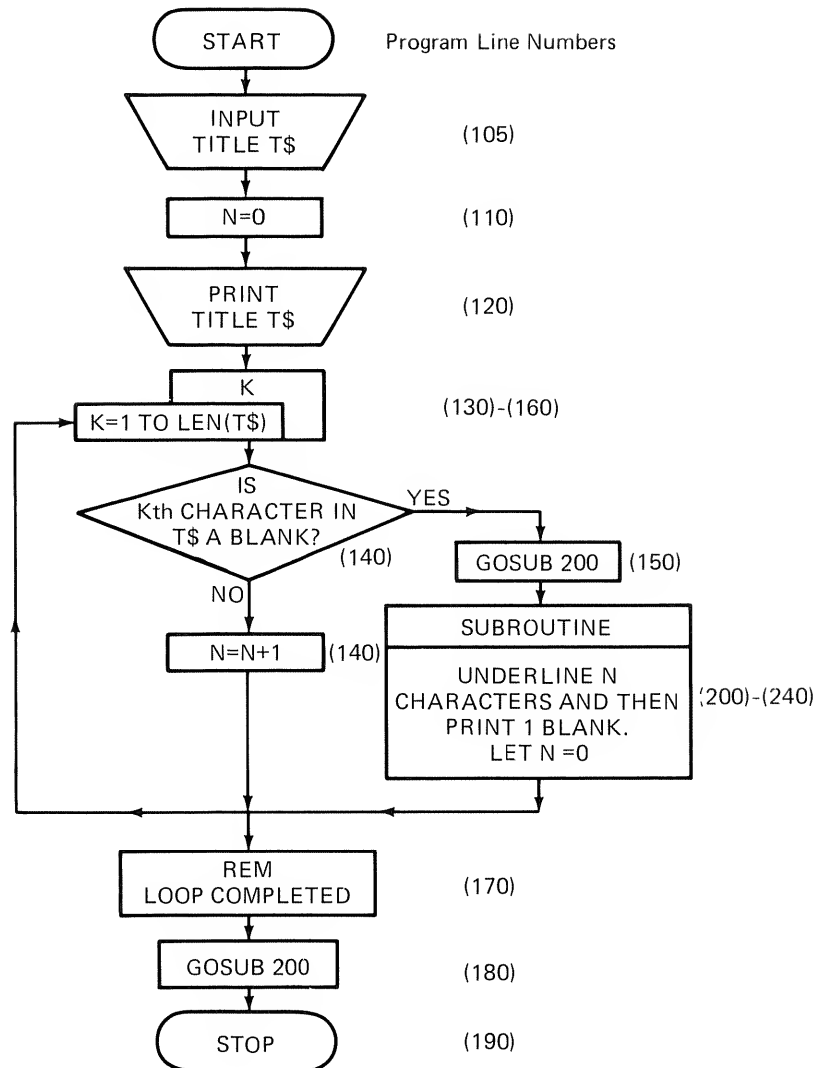


FIGURE 10.1 Underlining a title.

```

60 DEC=DEC+VAL(D$)*2^(N-1)
70 NEXT
80 PRINT "DECIMAL EQUIVALENT";DEC

```

```

RUN
BINARY NUMBER? 11
DECIMAL EQUIVALENT 3

```

The binary number is entered as a string of zeros and ones. Each character in the string, that is, each zero and each one, is converted to a number using the `VAL` function in line 60. It is then multiplied by the proper place value. The place values are  $2^0$ ,  $2^1$ ,  $2^2$ , and so on, for successive digits starting at the right end of the string `B$`. The sum in line 60 yields the decimal equivalent `DEC` of the binary number `B$`.

The `INKEY$` function is used to receive information, one character at a time, from the keyboard. The ENTER key need not be pressed to com-

plete the data entry. This is in contrast to entering data with the INPUT statement. With INKEY\$ the single character is automatically processed once the key is pressed. It is, however, not displayed on the screen. When the INKEY\$ is encountered, the keyboard is examined to determine if a key has been pressed. If no key has been pressed, the null character is assumed. The following short program places a single character at the top corner of the screen.

```
10 CLS
20 PRINT @ 0, INKEY$: GOTO 20
```

We execute the program and notice that the character corresponding to whichever key we press appears on the screen. To stop execution, press BREAK to escape from the infinite loop in line 20.

We now edit line 20 and take out the GOTO 20.

```
10 CLS
20 PRINT @ 0, INKEY$
```

When RUN is entered, the READY appears immediately, indicating completion of execution. The INKEY\$ needs to be placed within a loop so that the keyboard is repeatedly scanned for an entry. If it is not placed within a loop, as in the above edited version of the program, the keyboard is scanned only once. If by that time no entry has been made, INKEY\$ is a null string.

**REMEMBER:** The INKEY\$ function makes it possible to enter data without pressing the ENTER key.



#### *Example: Character String Entry Routine*

Any string up to 255 characters in length can be entered with this routine. There is no restriction on the type of characters entered; digits, letters, as well as symbols are permissible. Once the / character is entered the string is complete. The ENTER key need not be pressed.

```
10 CLEAR 1000: CLS
20 PRINT "ENTER ANY CHARACTER STRING"
30 FOR I=1 TO 255
40 Q$=INKEY$: IF Q$="" THEN 40
50 IF Q$="/" THEN 70 ELSE A$=A$+Q$
60 NEXT I
70 PRINT: PRINT "THE STRING YOU ENTERED IS"
80 PRINT A$
```

```
RUN
ENTER ANY CHARACTER STRING
```

```
THE STRING YOU ENTERED IS
HE SAID "PLEASE DON'T EAT THE DAISIES"
```

In this case we entered a string containing quotes and an apostrophe. They are all part of the string A\$.



### Example: Shoot the M's, A Video Game

The INKEY\$ function is particularly useful in writing programs to play video games. These games usually require quite sophisticated programming. Shoot the M's is a simple game for preschoolers. It requires no skill and only luck. The object of the game is to eliminate the M's from the screen by pressing the space bar. Once all the M's have disappeared, press the BREAK key and PRINT L to find out how many shots it took.

```

10 CLS: RANDOM
20 FOR K=6 TO 10
30 PRINT @ 64*K+26, "M M"
40 NEXT K
50 L=0
60 Q$=INKEY$: IF Q$="" THEN 60
70 L=L+1
80 PRINT @ RND(1000), Q$
90 GOTO 60

```

The loop of lines 20 to 40 displays on the screen a column consisting of five rows of the character string "M M". The counter L keeps track of the number of shots we take. Each shot is one INKEY\$ entry, which equals the character string Q\$. This entry is made by pressing the space bar. Q\$ is therefore a blank character string. It is printed in line 80 at a randomly determined print position. Eventually, the blanks replace all the M's and the game is over. At that point, press BREAK followed by PRINT L to display the number of shots taken to displace the M's.



### Example: Shoot the Duck

This game is a more sophisticated video game that requires some skill. A duck moves across the top of the screen. A gun, located at the lower edge of the screen, is fired at the duck. The up arrow key activates the gun, and the message \*BOOM\* appears on the screen in the event of a hit.

```

10 REM SHOOT THE DUCK
11 CLS: PRINT , "INSTRUCTIONS"
13 PRINT "UP ARROW KEY FIRES GUN"
14 PRINT: PRINT "***** GOOD LUCK *****"
15 INPUT "PRESS 'ENTER' WHEN READY"; Q$
16 T$=CHR$(128)+CHR$(191)+CHR$(191)+CHR$(176)
17 T$=T$+CHR$(253)+CHR$(190)+CHR$(189)+CHR$(254)
18 T$=T$+CHR$(142)+CHR$(140)+CHR$(255)
19 REM INITIALIZE POSITION OF GUN
20 CLS: SET(63,47)
23 REM MOVE DUCK ACROSS SCREEN ALONG Y=0
24 REM VARIABLE T$ IS THE DUCK
25 FOR K=1 TO 60: PRINT @ K, T$
26 REM CHECK FOR KEYBOARD ENTRY
30 C$=INKEY$: IF C$="" THEN 50
35 C=ASC(C$)
36 REM Y IS Y-POSITION OF BULLET
37 REM BULLET IS AT X=63
38 REM CHECK IF GUN JUST FIRED
39 ' AND BULLET NOT IN MOTION
40 REM Y IS SET TO 47 WHEN GUN IS FIRED
42 IF Y<=0 AND C=91 LET Y=47
50 SET(63,47) ' SET GUN POSITION

```

```

59 REM CHECK IF BULLET IN MOTION
60 IF Y<=0 THEN 90
70 REM BULLET IS AT POSITION (63,Y)
75 RESET(63,Y): Y=Y-4: IF Y<=0 GOTO 90
79 REM POINT(63,Y)=-1 WHEN BULLET HITS DUCK
80 IF POINT(63,Y)=0 SET(63,Y): GOTO 90
85 CLS: PRINT @ K, "*BOOM*"
86 REM TIME DELAY FOR *BOOM* DISPLAY
87 FOR L=1 TO 250: NEXT L
89 GOTO 20
90 NEXT K: GOTO 25

```

The program contains many REM statements that explain the code. In addition, the following comments may be helpful:

*Lines 16 to 18:* Variable T\$ consists of several graphic characters that together assume the shape of a duck.

*Line 25:* The duck moves one step at a time across the screen; the end of the FOR-NEXT loop is at line 90. As the duck moves across the screen (along Y = 0), its horizontal position is specified by the value of K.

*Line 75:* The bullet moves straight up. Its vertical position Y therefore continuously changes as the value of Y decreases from 47 to 0. The horizontal position of the bullet is unchanged and equals the horizontal position of the gun at X = 63. The bullet's position is RESET before each move. Its path therefore appears as a sequence of dots and not as a line.

*Line 80:* If the bullet reaches a position that is set by the presence of the duck, that is, for which POINT(63,Y) = -1, then the bullet is colliding with the duck and the message \*BOOM\* appears.

*Line 87:* The time delay leaves the message \*BOOM\* on the screen for a short period of time.

*Line 89:* The game starts over after the time delay of line 87. To stop execution, press BREAK.

The game illustrates video game programming. Several refinements could be included to improve the game. The duck could be made to move back and forth, and a scoreboard could be added. The game may be more fun if the position of the gun is controlled by the player and if a timer were incorporated to score the number of hits within a specific time interval.



### Example: Etch-a-Sketch

This popular children's toy is used to draw a pattern of lines—a graphic display. We simulate this toy on the computer.

```

10 REM ETCH-A-SKETCH
15 X=63: Y=23: CLS
20 ON ERROR GOTO 80
29 REM BLINKING BLOCK
30 RESET(X,Y): SET(X,Y)
35 RESET(X,Y): SET(X,Y)
40 C$=INKEY$: IF C$="" THEN 30
50 C=ASC(C$)
54 REM RIGHT ARROW MOVE?

```

### COMMENTS

The starting point is at mid-screen.

A sequence of RESET and SET produces blinking.

The next move is entered.

ASCII code of right arrow key is 9;

|                               |                                    |
|-------------------------------|------------------------------------|
| 55 IF C=9 LET X=X+1           | increment 1 position to right.     |
| 59 REM LEFT ARROW MOVE?       | ASCII code of left arrow key is 8; |
| 60 IF C=8 LET X=X-1           | increment 1 position to left.      |
| 64 REM UP ARROW MOVE?         | ASCII code for up arrow key is 91; |
| 65 IF C=91 LET Y=Y-1          | increment 1 position up.           |
| 69 REM DOWN ARROW MOVE?       | ASCII code for down arrow key is   |
| 70 IF C=10 LET Y=Y+1          | 10; increment 1 position down.     |
| 75 SET(X,Y): GOTO 40          | Draw line to new position.         |
| 80 REM ERROR ROUTINE          |                                    |
| 84 REM AVOID GOING OFF SCREEN | Remember, Y = 0 is along top of    |
|                               | screen.                            |
| 85 IF X>127 LET X=127         | Do not allow X to exceed 127 or be |
| 90 IF X<0 LET X=0             | less than 0.                       |
| 93 IF Y>47 LET Y=47           | Do not allow Y to exceed 47 or be  |
| 95 IF Y<0 LET Y=0             | less than 0.                       |
| 98 RESUME 40                  | To stop execution, press BREAK.    |

The sketch starts out with a small blinking block at mid-screen ( $X = 63, Y = 23$ ). From there, depending on which key is pressed, a line is drawn sideways, up, or down. The `INKEY$` function is used in line 40 to specify the direction in which the line is to be drawn. The ASCII code of the key pressed is determined in line 50 and then tested in lines 55 to 70 to determine the direction in which the line is to proceed. The new line segment is drawn in line 75 using the `SET` function. The error routine (lines 85 to 98) ensures that execution is not interrupted when the cursor attempts to go off screen.

There are several ways in which the program can be modified: the starting point of the sketch may be placed at a position other than mid-screen or larger line segments may be drawn. For example, it may be desirable to make the line segments in the right-left directions three times as large as the up-down segments. Another possibility is to introduce an additional key that will draw along the diagonal. This will give the "artist" added flexibility. Can you implement any of these changes?

## EXERCISES 13

- Before executing the instructions, fill in the anticipated display and compare it with the actual display.

| Instruction                | Anticipated Display | Display |
|----------------------------|---------------------|---------|
| a. PRINT ASC("B")          |                     |         |
| b. PRINT ASC("BOY")        |                     |         |
| c. PRINT ASC(5)            |                     |         |
| d. PRINT ASC(FIVE)         |                     |         |
| e. PRINT ASC("S")          |                     |         |
| f. PRINT ASC(" ")          |                     |         |
| g. PRINT ASC("")           |                     |         |
| h. PRINT CHR\$(48)         |                     |         |
| i. PRINT CHR\$(65)         |                     |         |
| j. PRINT CHR\$(65+32)      |                     |         |
| k. PRINT CHR\$(94+32)      |                     |         |
| l. PRINT CHR\$(23)         |                     |         |
| m. CLEAR                   |                     |         |
| n. PRINT ASC(CHR\$(70))    |                     |         |
| o. PRINT CHR\$(ASC("TRS")) |                     |         |
| p. Q\$=CHR\$(34)           |                     |         |
| q. PRINT Q\$+"TRS-80"+Q\$  |                     |         |



2. When filling in the anticipated displays, be sure to account for leading and trailing blanks.

| Instruction                             | Anticipated Display | Display |
|-----------------------------------------|---------------------|---------|
| a. Q\$="TO BE OR NOT TO BE"             |                     |         |
| b. PRINT LEN(A\$)                       |                     |         |
| c. PRINT LEN(Q\$)                       |                     |         |
| d. PRINT LEFT\$(Q\$, 5)                 |                     |         |
| e. PRINT LEFT\$(Q\$, 0)                 |                     |         |
| f. PRINT LEFT\$(Q\$, 20)                |                     |         |
| g. PRINT LEFT\$(Q\$, LEN(Q\$))          |                     |         |
| h. PRINT RIGHT\$(Q\$, 5)                |                     |         |
| i. PRINT RIGHT\$(Q\$, 5)+LEFT\$(Q\$, 5) |                     |         |

3. Which of the following expressions are equivalent? Assume:

A\$= "I AM EARLY"  
B\$ = "I AM LATE"

- PRINT LEFT\$(A\$, 4)
- PRINT MID\$(A\$, 6, 1)
- PRINT RIGHT\$(A\$, 1)
- PRINT LEFT\$(B\$, 5)
- PRINT A\$
- PRINT LEN(A\$)
- PRINT LEN(LEFT\$(A\$, 4))
- PRINT MID\$(B\$, 1, 4)
- PRINT MID\$(A\$, LEN(A\$), 1)
- PRINT LEN(RIGHT\$(B\$, 4))
- PRINT LEN(B\$)
- PRINT MID\$(RIGHT\$(A\$, 6), 6, 1)
- PRINT LEFT\$(B\$, 5)+RIGHT\$(A\$, 5)

| Instruction                   | Anticipated Display | Display |
|-------------------------------|---------------------|---------|
| a. PRINT FRE("50")            |                     |         |
| b. PRINT STRING\$(40, "-")    |                     |         |
| c. PRINT STRING\$(80, "-")    |                     |         |
| d. PRINT STRING\$(0, "-")     |                     |         |
| e. PRINT STRING\$(40, "-")    |                     |         |
| f. PRINT STRING\$(4, 5, "-")  |                     |         |
| g. PRINT STRING\$(4, "ABC")   |                     |         |
| h. PRINT FRE("-")             |                     |         |
| i. CLEAR 400                  |                     |         |
| j. PRINT FRE("S")             |                     |         |
| k. PRINT STRING\$(255, "S")   |                     |         |
| l. PRINT STRING\$(255, "SSS") |                     |         |
| m. PRINT STRING\$(256, "END") |                     |         |
| n. A\$="FUN"                  |                     |         |
| o. PRINT FRE("A\$")           |                     |         |
| p. PRINT MEM                  |                     |         |
| q. PRINT VAL("12")-12         |                     |         |
| r. PRINT "A"+STR\$(1)         |                     |         |
| s. PRINT LEN(STR\$(123))      |                     |         |

| Instruction                          | Anticipated Display | Display |
|--------------------------------------|---------------------|---------|
| t. PRINT LEN(STR\$(123))             | _____               | _____   |
| u. PRINT VAL(10)                     | _____               | _____   |
| v. PRINT STR\$(VAL("12"))            | _____               | _____   |
| w. 10 PRINT @ 1000; INKEY\$; GOTO 10 | _____               | _____   |
| x. RUN                               | _____               | _____   |

5. What do the following programs accomplish?

- |                                                                                                                                                                                                                      |                                                                                                                                                        |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------|
| <p>a. 10 FOR K=1 TO 3<br/> 20 C\$=INKEY\$: IF C\$="" THEN 20<br/> 30 T\$=T\$+C\$: NEXT K<br/> 40 CLS: PRINT @ 500; T\$<br/> 50 GOTO 50</p> <p>c. 10 INPUT N<br/> 20 PRINT N; "COMPUTER"; CHR\$(ABS(SGN(N-1)*83))</p> | <p>b. 10 CLS: K=65<br/> 20 FOR A=0 TO 25<br/> 30 FOR B=0 TO 63<br/> 40 PRINT CHR\$(K);<br/> 50 NEXT B<br/> 60 K=K+1<br/> 70 NEXT A<br/> 80 GOTO 80</p> |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------|

6. a. Write a program to input a text and output the number of E's occurring in the text.  
b. Modify the program of part a to output the number of times each vowel occurs in the text.  
c. Modify the program of part (a) to display the number of times the ending ING appears in the text.
7. Write a program to input a coded message along with its code value and output the decoded message. Before writing the program, review the example Coding a Message presented in this chapter.
8. An imperfect palindrome is a statement that reads the same forward or backward after all nonalphabetic characters have been removed. Write a program to input a text and test if it is an imperfect palindrome. Check your program on

```
MADAM I'M ADAM
EGAD A BASE TONE DENOTES A BAD AGE
```

*Hint:* Determine the ASCII code of each of the string's characters and test if it falls in the range 65 to 90 inclusive; if it does not, the character is dropped since it is not a letter.

9. Write a program to input the names of N different states and list them in a right-aligned format. For example,

```
CONNECTICUT
RHODE ISLAND
 MAINE
MASSACHUSETTS
NEW HAMPSHIRE
 VERNONT
```

*Hint:* Use the TAB and LEN functions in a PRINT statement.

10. Determine the longest word in a sentence.
11. Suppose the variable W\$ is a character string containing only letters. Determine the number of letters in W\$ that are also in "TOBY".
12. Program the computer to play the game of BUZZ. In this game we count from 1 to

100, but for any number containing the digit 7, display BUZZ instead of displaying the number. For numbers that contain a 7 and are divisible by 7 display BUZZ BUZZ.

13. **Merge** two lists of names that are in alphabetic order into a single list of names also in alphabetic order.
14. Find every two-digit number that equals the sum of the squares of its digits.

# appendix I error messages

The following table summarizes the Level II BASIC error messages and includes a brief description of their cause and corrective action.

| Error Message | Error Code Number | Cause (Corrective Action)                                                                                                              |
|---------------|-------------------|----------------------------------------------------------------------------------------------------------------------------------------|
| NF            | 1                 | FOR statement missing, or I and K in NEXT I,K out of order. Add FOR statement, or delete NEXT statement.                               |
| SN            | 2                 | Syntax error. Incorrectly formed statement; for example, error in punctuation or unmatched parentheses.                                |
| RG            | 3                 | RETURN without GOSUB. GOSUB must be executed before RETURN. Delete RETURN statement or add a GOSUB statement.                          |
| OD            | 4                 | Out of data. Insufficient data in INPUT # or READ-DATA.                                                                                |
| FC            | 5                 | Illegal use of a function. Adhere to the proper format of the function; for example, cannot take the square root of a negative number. |
| OV            | 6                 | Overflow. Value exceeds allowable maximum; for example, an integer cannot exceed 32767.                                                |
| OM            | 7                 | Out of memory. Memory capacity of the computer exceeded. Cut back on DIM, REM, and other statements.                                   |
| UL            | 8                 | Undefined line. Reference made to a missing line number. Add the line to which execution is to branch.                                 |
| BS            | 9                 | Subscript out of range. Modify the DIM statement to include the desired subscript.                                                     |
| DD            | 10                | An array is dimensioned twice. Specify array's DIM only once.                                                                          |
| /0            | 11                | Division by zero. Avoid zero in the denominator.                                                                                       |
| ID            | 12                | Illegal use of INPUT; line number missing.                                                                                             |
| TM            | 13                | Mismatch of variable types; for example, a string variable is assigned to a numeric variable, or vice versa.                           |
| OS            | 14                | Out of string space. Increase N in CLEAR N.                                                                                            |
| LS            | 15                | String too long. Maximum string length is 255 characters.                                                                              |
| ST            | 16                | String operation too complicated. Break up into shorter steps.                                                                         |
| CN            | 17                | CONTINUE statement cannot be executed; for example, after an END statement, or after editing.                                          |
| NR            | 18                | RESUME statement is missing. Add a RESUME at the end of the error-trapping routine.                                                    |
| RW            | 19                | RESUME encountered without an ON ERROR GOTO. Check for missing ON ERROR GOTO.                                                          |

| Error<br>Message | Error<br>Code<br>Number | Cause (Corrective Action)                                                                                |
|------------------|-------------------------|----------------------------------------------------------------------------------------------------------|
| UE               | 20                      | Invalid error code number used with <b>ERROR</b> statement. Check list of error code numbers.            |
| MO               | 21                      | The operation is missing an operand. Add the operand.                                                    |
| FD               | 22                      | File data are not acceptable. Data on tape are not compatible.                                           |
| L3               | 23                      | This BASIC statement is only available when the computer's mini disk is connected through the interface. |

## appendix II reserved words

None of the following words can be used inside a variable name. A syntax error will occur if these words are used as variables. However, all words ending with the symbol \$ (for example CHR\$) may be used as legal variables when the \$ is dropped. Therefore, CHR is a legal variable. Some of the words listed below have no function in Level II BASIC; they are reserved for Level II Disk BASIC.

|        |         |        |          |
|--------|---------|--------|----------|
| @      | ELSE    | LOC    | RESTORE  |
| ABS    | END     | LOF    | RESUME   |
| AND    | EOF     | LOG    | RETURN   |
| ASC    | ERL     | LSET   | RIGHT\$  |
| ATN    | ERR     | MEM    | RND      |
| AUTO   | ERROR   | MERGE  | RSET     |
| CDBL   | EXP     | MID\$  | RUN      |
| CHR\$  | FIELD   | MKD\$  | SAVE     |
| CINT   | FIX     | MKI\$  | SET      |
| CLEAR  | FN      | MKS\$  | SGN      |
| CLOCK  | FOR     | NAME   | SIN      |
| CLOSE  | FORMAT  | NEW    | SQR      |
| CLS    | FRE     | NEXT   | STEP     |
| CMD    | FREE    | NOT    | STOP     |
| CONT   | GET     | ON     | STRING\$ |
| COS    | GOSUB   | OPEN   | STR\$    |
| CSNG   | GOTO    | OR     | SYSTEM   |
| CVD    | IF      | OUT    | TAB      |
| CVI    | INKEY\$ | PEEK   | TAN      |
| CVS    | INP     | POINT  | THEN     |
| DATA   | INPUT   | POKE   | TIMES\$  |
| DEFDBL | INSTR   | POS    | TO       |
| DEFFN  | INT     | POSN   | TROFF    |
| DEFINT | KILL    | PRINT  | TRON     |
| DEFSNG | LEFT\$  | PUT    | USING    |
| DEFSTR | LEN     | RANDOM | USR      |
| DEFUSR | LET     | READ   | VAL      |
| DELETE | LINE    | REM    | VARPTR   |
| DIM    | LIST    | RENAME | VERIFY   |
| EDIT   | LOAD    | RESET  |          |

# appendix BASIC glossary

The following glossary provides short descriptions of Level II BASIC commands, instructions, and functions. Most of the terms are described in detail within the text and can be located readily through the index. In addition, the glossary includes some frequently encountered programming and data-processing terms.

|                               |                                                                                                                                             |
|-------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------|
| <i>Abbreviations</i>          | Level II BASIC allows for only three abbreviations: ? for PRINT, ' for REM, . for last line entered, edited, or in which an error occurred. |
| <b>ABS(X)</b>                 | A library function to determine the absolute value of X. The absolute value of X is always positive; for example, ABS(5)=5, and ABS(-5)=5.  |
| <i>Access time</i>            | The time interval between the instant at which data are called for from storage and the instant delivery begins.                            |
| <i>Address</i>                | A location in memory where a byte is stored.                                                                                                |
| <i>Algorithm</i>              | A step-by-step procedure for solving a specific problem, or for performing a specific task.                                                 |
| <i>Alphanumeric character</i> | Any letter, digit, or special symbol.                                                                                                       |
| <b>AND</b>                    | A logical operation; -1 AND -1 = -1; 0 AND -1 = 0; 0 AND 0 = 0.                                                                             |
| <i>Argument</i>               | The expression on which a function operates to yield a specific result; for example, 5 is the argument in SQR(5).                           |
| <i>Arithmetic operators</i>   | Perform arithmetic with numeric variables: addition (+), subtraction (-), multiplication (*), division (/), exponentiation ( $\uparrow$ ).  |
| <i>Array</i>                  | A subscripted variable; an ordered list of numbers or strings.                                                                              |
| <b>ASC(string)</b>            | Returns the ASCII code of the first character of the string. Performs the inverse of the CHR\$ function.                                    |
| <i>ASCII code</i>             | American Standard Code for Information Interchange; in Level II BASIC the codes correspond to numbers 0 to 255.                             |
| <i>Assembler</i>              | Converts a symbolic language program into machine language.                                                                                 |
| <i>Assignment</i>             | The procedure by which the value of a variable is specified. Assignment is indicated by the equal sign; for example, X=5.                   |
| <b>ATN(X)</b>                 | A library function to determine the arctangent of X. Displays the angle (in radians) whose tangent is X.                                    |
| <i>AUTO</i>                   | A command for automatic line numbering.                                                                                                     |
| <i>BASIC</i>                  | Beginner's All-purpose Symbolic Instruction Code; one of many computer languages.                                                           |
| <i>Baud</i>                   | Speed in which information is transferred; measured in bits per second.                                                                     |

|                             |                                                                                                                                                                                                                          |
|-----------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Binary number</b>        | A number in the base 2 number system consisting of the digits 0 and 1.                                                                                                                                                   |
| <b>Bit</b>                  | Binary digIT; a single digit in the binary number system; i.e., a binary 1 or a binary 0.                                                                                                                                |
| <b>Branching</b>            | See <i>transfer statements</i> .                                                                                                                                                                                         |
| <b>BREAK key</b>            | Stops execution; to resume type CONT. Also used to escape the AUTO command.                                                                                                                                              |
| <b>Built-In function</b>    | See <i>library functions</i> .                                                                                                                                                                                           |
| <b>Byte</b>                 | Smallest unit of memory; consists of 8 bits.                                                                                                                                                                             |
| <b>CBDL(X)</b>              | A library function to present X in double precision.                                                                                                                                                                     |
| <b>Chaining</b>             | Forming a multiple statement line. Saves memory space. Colons are used between statements to chain them together into one line.                                                                                          |
| <b>CHR\$(N)</b>             | Returns the character that has N as its ASCII code. The argument N may be any expression whose numerical value is 0 to 255. Performs the inverse of the ASC function.                                                    |
| <b>CINT(X)</b>              | A library function to determine the largest integer not greater than X. The argument X must be between -32768 and 32767; for example, CINT(2.2)=2, and CINT(-2.12)=-3.                                                   |
| <b>CLEAR</b>                | A command to set all numeric variables to zero and all string variables to null.                                                                                                                                         |
| <b>CLEAR key</b>            | Clears screen; returns cursor to first line; switches from 32 to 64 characters per line format.                                                                                                                          |
| <b>CLEAR N</b>              | A command to reserve N bytes of memory for strings. In addition, the command sets all numeric variables to zero and string variables to null. When the computer is first turned on, 50 bytes are automatically reserved. |
| <b>CLOAD</b>                | Command to load the very next program stored on the cassette into the computer.                                                                                                                                          |
| <b>CLOAD "NAME"</b>         | Command to load the program NAME from cassette into the computer. Only the first character of NAME is recognized by the computer.                                                                                        |
| <b>CLOAD? "NAME"</b>        | Command to compare a program on cassette with the program in the computer. Useful in checking whether a program was properly stored on cassette.                                                                         |
| <b>CLS</b>                  | A statement to clear the screen and move the cursor to the top left.                                                                                                                                                     |
| <b>Coding</b>               | Writing a computer program.                                                                                                                                                                                              |
| <b>Colon (:) key</b>        | Used to chain statements together; for example, 10 FOR I=1 TO 5: PRINT I: NEXT.                                                                                                                                          |
| <b>Comma</b>                | Causes the items of a list in a PRINT statement to be displayed in successive zones across the screen; for example, PRINT A,B,C.                                                                                         |
| <b>Command mode</b>         | Computer responds to commands upon entry. When in this mode, the >_ is displayed. In the command mode, distinction is made among immediate and programming modes.                                                        |
| <b>Compiler</b>             | Converts a high-level language program into machine language.                                                                                                                                                            |
| <b>Computer program</b>     | See <i>program</i> .                                                                                                                                                                                                     |
| <b>Concatenate</b>          | String together two or more string variables; for example, "AB" + "C" gives "ABC". The plus sign identifies concatenation.                                                                                               |
| <b>Conditional transfer</b> | A statement that transfers execution to a specific line in a program depending on whether a certain condition is met.                                                                                                    |
| <b>CONT</b>                 | Command to continue execution after it has been stopped with the BREAK key or with a STOP statement. BREAK and CONT are useful in debugging.                                                                             |
| <b>COS(X)</b>               | A library function to determine the cosine of X. The argument X must be in radians.                                                                                                                                      |
| <b>CSAVE "NAME"</b>         | Command to store program currently in the computer on tape. To transfer the program back from cassette to computer, use CLOAD "NAME".                                                                                    |



|                                  |                                                                                                                                                                                                                                                |
|----------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>CSNG(X)</b>                   | A library function to form the single-precision version of X; for example, CSNG (1.234567890) is 1.23457.                                                                                                                                      |
| <i>Cursor</i>                    | Pointer on the screen indicating position in which next typed entry will be made.                                                                                                                                                              |
| <b>DATA</b>                      | A statement that stores data in one or more lines within a program. These data can be read with a READ statement; for example, 10 DATA 10,20,"ERIC".                                                                                           |
| <i>Data validity check</i>       | Giving the operator an opportunity to check the data just entered and to make corrections before data are processed.                                                                                                                           |
| <i>Debugging</i>                 | The process of locating and removing errors from a program.                                                                                                                                                                                    |
| <b>DEFDBL letters</b>            | Variables beginning with the specified letters are declared as double-precision variables.                                                                                                                                                     |
| <b>DEFINT letters</b>            | Variables beginning with the specified letters are declared as integer variables. Saves memory and executes faster.                                                                                                                            |
| <b>DEFSNG letters</b>            | Variables beginning with the specified letters are declared as single-precision variables.                                                                                                                                                     |
| <b>DEFSTR letters</b>            | Variables beginning with the specified letters are declared as string variables.                                                                                                                                                               |
| <b>DELETE</b>                    | Command to erase a line or several lines of a program in memory; useful in editing; for example, DELETE 50, DELETE -50, or DELETE 50-80.                                                                                                       |
| <i>Device number</i>             | In the presence of two cassette drives, it is necessary to specify which cassette drive is to be accessed; for example, CLOAD #-1,"F" or PRINT#-2,A,B.                                                                                         |
| <b>DIM</b>                       | Reserves memory space for subscripted variables. Specifies the range of each subscript. If no DIM statement is used, a range of 11 (subscripts 0 to 10) is allowed as the dimension of each array; for example, 10 DIM A(10,10) is not needed. |
| <i>Double-precision variable</i> | Sixteen significant figures; for example, A#=1.234567890123456.                                                                                                                                                                                |
| <i>Dual cassettes</i>            | Using two cassettes on line.                                                                                                                                                                                                                   |
| <i>Dummy argument</i>            | An argument that can take on any value whatsoever and produce the same result.                                                                                                                                                                 |
| <i>Edit mode</i>                 | Add, change, or delete lines in a program. Enter mode through EDIT N (N = line number). Escape edit mode by pressing Q.                                                                                                                        |
| <b>EDIT N</b>                    | Command to display line number N and switch to edit mode.                                                                                                                                                                                      |
| <b>END</b>                       | This statement terminates execution without a BREAK. END is optional. In its absence, execution stops when the last statement of the program is executed.                                                                                      |
| <i>ENTER key</i>                 | Press to communicate an instruction to the computer.                                                                                                                                                                                           |
| <b>ERL</b>                       | Function returns the line number at which an error has occurred. It equals zero if no error has occurred. If an error occurs in immediate mode, ERL=65536. Useful within an error-handling routine.                                            |
| <b>ERR</b>                       | Function returns the (error code -1)*2 of the error that has just occurred. Useful within an error-handling routine.                                                                                                                           |
| <b>ERR/2+1</b>                   | Function returns the error code of the error that has just occurred. Useful within an error-handling routine.                                                                                                                                  |
| <b>ERROR code</b>                | This statement causes the computer to proceed exactly as if the error corresponding to the specified code has occurred. Useful in testing an ON ERROR GOTO routine.                                                                            |
| <i>Error message</i>             | Diagnostic information given by the computer about an error in the program; see Appendix 1.                                                                                                                                                    |
| <i>Error N</i>                   | Every error has a code number N. There are a total of 23 error codes numbered 1 to 23.                                                                                                                                                         |
| <i>Execute mode</i>              | Computer executes a program; mode entered by typing in RUN.                                                                                                                                                                                    |

|                                |                                                                                                                                                                                                            |
|--------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Execution</b>               | Programs are executed to yield results.                                                                                                                                                                    |
| <b>Execution sequence</b>      | Unless otherwise specified the execution of a program is sequential by line number, starting with the first statement of the program.                                                                      |
| <b>Expansion interface</b>     | Allows the use of additional devices: a second cassette, a printer, up to four mini disk drives, and up to 48K bytes of RAM.                                                                               |
| <b>Exponentiation</b>          | $A \uparrow B$ ; A to the power B; for example, $2 \uparrow 3 = 8$ .                                                                                                                                       |
| <b>EXP(X)</b>                  | A library function to determine $e^x$ ; for example, $\text{EXP}(1) = 2.71828$ .                                                                                                                           |
| <b>EXTRA IGNORED</b>           | Computer's response to entry of more data than required by INPUT. Execution continues without interruption.                                                                                                |
| <b>File</b>                    | An organized collection of information.                                                                                                                                                                    |
| <b>FIX(X)</b>                  | A library function to truncate X at the decimal point; for example, $\text{FIX}(3.1415) = 3$ .                                                                                                             |
| <b>Flag</b>                    | A special identifier; for example, it may identify the last data entry or the last record in a file.                                                                                                       |
| <b>Flowchart</b>               | A grid of boxes interconnected by arrows, which shows the overall structure of a computer program.                                                                                                         |
| <b>FOR</b>                     | The first statement in a FOR-NEXT loop.                                                                                                                                                                    |
| <b>FOR-NEXT loop</b>           | The first statement of the loop is the FOR statement; the NEXT statement is the last. Together the sequence of statements constitutes a loop that is executed a specified number of times.                 |
| <b>FRE(string)</b>             | Returns the available string storage space; the argument string may be any (dummy) string variable or constant. Available space is related to CLEAR N.                                                     |
| <b>GOSUB N</b>                 | Unconditional transfer to the subroutine beginning at line N.                                                                                                                                              |
| <b>GOTO N</b>                  | Unconditional transfer to line N. In immediate mode (no line number), this statement will start execution at line N. Unlike the RUN command, it does not initialize all variables to zero and null.        |
| <b>Graphic block</b>           | The video display is divided into 1024 graphic blocks: 128 horizontal and 48 vertical blocks. These can be addressed through the graphic functions. The point 0,0 is at the top left corner of the screen. |
| <b>Halt execution</b>          | Press SHIFT @ to freeze the display. To resume execution, press any key.                                                                                                                                   |
| <b>Hanging comma</b>           | Suppresses the line feed. Successive PRINT statements display in successive zones on the same line; for example, 50 PRINT A,                                                                               |
| <b>Hanging semicolon</b>       | Suppresses the line feed. Successive PRINT statements display on the same line; for example, 50 PRINT A;                                                                                                   |
| <b>Hexadecimal number</b>      | A number in base 16.                                                                                                                                                                                       |
| <b>Hierarchy of operations</b> | Order of priority for performing numeric and string operations.                                                                                                                                            |
| <b>High-level language</b>     | A computer language easy for humans to use. A program in such a language needs to be translated to machine language before it can be executed.                                                             |
| <b>IF-THEN-ELSE</b>            | A statement to test an expression. Depending on whether it is true or false, execution transfers to different lines within a program. The IF is a conditional transfer statement.                          |
| <b>Immediate mode</b>          | A form of the command mode; no line numbers are used; also called calculator mode.                                                                                                                         |
| <b>INKEY\$</b>                 | Monitors the keyboard over a short interval and displays the key pressed within the interval. If no key pressed, returns the null (empty) string.                                                          |
| <b>INP(P)</b>                  | Returns the current value from the specified port P. There are 256 ports numbered 0 through 255. Requires an expansion interface.                                                                          |

|                                 |                                                                                                                                                                                                                                                                        |
|---------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>INPUT "MESSAGE";<br/>A,B</b> | Displays the message (if any) and interrupts execution so that values may be entered from the keyboard for A and B.                                                                                                                                                    |
| <b>INPUT#-1,A,B</b>             | Input the values of A and B from the cassette. The input list must be identical to the PRINT#-1,A,B that created the tape.                                                                                                                                             |
| <b>Integer arithmetic</b>       | Arithmetic with integer variables. It is faster than single- or double-precision arithmetic. Results of arithmetic are integers; for example, $3/2=1$ .                                                                                                                |
| <b>Integer variable</b>         | Whole numbers between -32768 and +32767 inclusive; for example, A%=1984. Integer variables take up less memory than other variables.                                                                                                                                   |
| <b>Interpreter</b>              | A program that takes the high-level language program and leads the computer through the steps necessary to execute it. An interpreter does not translate the source program into an object program. It merely interprets the code so that the computer can execute it. |
| <b>INT(X)</b>                   | A library function to determine the largest integer not greater than X; for example, INT(32800.5)=32800, and INT(-32800.5)=-32801.                                                                                                                                     |
| <b>← Key</b>                    | Backspaces the cursor and erases characters.                                                                                                                                                                                                                           |
| <b>↓ Key</b>                    | Moves the cursor down to the next line.                                                                                                                                                                                                                                |
| <b>→ Key</b>                    | Moves the cursor to the next tab position. Tab positions are at 0, 8, 16, 24, 32, 48, and 56.                                                                                                                                                                          |
| <b>Keyboard rollover</b>        | You can press a second key before releasing the first key.                                                                                                                                                                                                             |
| <b>Language errors</b>          | Errors detected by the computer. The computer checks each line of code and lists errors in the form of diagnostic messages.                                                                                                                                            |
| <b>Left to right rule</b>       | BASIC statements are executed from left to right and are in addition subject to a hierarchy of operations.                                                                                                                                                             |
| <b>LEFT\$(string,N)</b>         | Isolates the first N characters of string.                                                                                                                                                                                                                             |
| <b>LEN(string)</b>              | Determines the number of characters including blanks in string.                                                                                                                                                                                                        |
| <b>LET</b>                      | An assignment statement. The LET is optional; i.e., 10 LET A=5 and 10 A=5 are identical.                                                                                                                                                                               |
| <b>Library functions</b>        | Functions to perform a variety of different tasks that have been programmed into the computer. These functions can be called upon within the program; for example, the function SQR(X) determines the square root of X.                                                |
| <b>Line length</b>              | A line within a program may be up to 255 characters in length.                                                                                                                                                                                                         |
| <b>Line number</b>              | Identifies a line in a program; any integer from 0 to 65529 is permitted.                                                                                                                                                                                              |
| <b>LIST</b>                     | Command to display the program currently in the computer. LIST 20 displays just line 20. Specific sections of the program are displayed by LIST-20, LIST20-, or LIST 20-40.                                                                                            |
| <b>LLIST</b>                    | Identical to LIST except lists on the line printer.                                                                                                                                                                                                                    |
| <b>Logical errors</b>           | Using an incorrect formula or an inappropriate function will yield incorrect results. Errors in logic are more difficult to isolate than language errors since the computer does not detect them and does not respond with diagnostic messages.                        |
| <b>Logical expression</b>       | An expression that is either true or false. The expression equals -1 if it is true and 0 if it is false.                                                                                                                                                               |
| <b>Logical operators</b>        | Operators that relate logical data and return logical results. Use with numeric and string variables (AND, NOT, OR).                                                                                                                                                   |
| <b>Logical variables</b>        | A variable whose value is -1 or 0. It is specified by means of a logical operation.                                                                                                                                                                                    |
| <b>LOG(X)</b>                   | A library function to determine the log to the base <i>e</i> of X.                                                                                                                                                                                                     |
| <b>Looping</b>                  | A repetitive process in which several lines within a program are executed a specified number of times.                                                                                                                                                                 |

|                                   |                                                                                                                                                                                                                    |
|-----------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>Low-level language</i>         | Machine language; computer can directly perform operations specified in machine language.                                                                                                                          |
| <b>LPRINT</b>                     | Identical to <b>PRINT</b> except prints on the line printer.                                                                                                                                                       |
| <i>Machine language</i>           | The language used by the computer; it is in binary.                                                                                                                                                                |
| <i>Main program</i>               | The portion of the computer program that does not include the subroutines. Access to the subroutines is gained from the main program.                                                                              |
| <i>Matrix</i>                     | A two-dimensional array; for example, $A(I,J)$ .                                                                                                                                                                   |
| <b>MEM</b>                        | Returns the number of unused and unprotected bytes in memory.                                                                                                                                                      |
| <i>Memory map</i>                 | Address locations of TRS-80 BASIC.                                                                                                                                                                                 |
| <b>MEMORY SIZE?</b>               | Appears on the screen when the computer is first turned on. Respond by pressing <b>ENTER</b> unless you wish to reserve some memory for a machine language program.                                                |
| <b>MID\$(string,M,N)</b>          | Isolates N characters of string, starting with the Mth character. If N is not specified, all the characters starting with the Mth are isolated.                                                                    |
| <i>Mini disk</i>                  | A storage device capable of storing up to 89,600 bytes. Has an access time much faster than tapes. Allows sequential and random access.                                                                            |
| <i>Monitor mode</i>               | Loads machine language programs.                                                                                                                                                                                   |
| <i>Nested loops</i>               | One loop is completely enclosed in another loop. Innermost loop is executed most rapidly.                                                                                                                          |
| <b>NEW</b>                        | Command to erase the entire program currently in the computer; sets all numeric variables to zero and all strings to null. Does not change the string space reserved by a previously executed <b>CLEAR N</b> .     |
| <b>NEXT</b>                       | The last statement of a <b>FOR-NEXT</b> loop.                                                                                                                                                                      |
| <b>NOT</b>                        | A logical operation; $\text{NOT } -1=0$ ; $\text{NOT } 0=-1$ .                                                                                                                                                     |
| <i>Null string</i>                | An empty string containing no characters. The <b>RUN</b> command sets all numeric variables to zero and all string variables to null.                                                                              |
| <i>Object program</i>             | The compiled version of a source program. It is in a form that can be executed directly.                                                                                                                           |
| <i>One-dimensional array</i>      | A subscripted variable with one subscript; for example, $A(I)$ .                                                                                                                                                   |
| <b>ON ERROR GOTO N</b>            | An unconditional transfer; when an error occurs, execution transfers to line N, where an error-trapping routine starts.                                                                                            |
| <b>ON N GOSUB N1,N2, N3 . . .</b> | A conditional transfer to one of several subroutines; if $N=1$ transfer to the subroutine beginning at line N1, if $N=2$ to line N2, and so on.                                                                    |
| <b>ON N GOTO N1,N2, N3 . . .</b>  | A conditional transfer; if $N=1$ execution transfers to line N1, if $N=2$ to line N2, and so on.                                                                                                                   |
| <i>Operators</i>                  | Perform arithmetic, logical, relational, and string operations; for example, addition or a less-than comparison.                                                                                                   |
| <b>OR</b>                         | A logical operation; $-1 \text{ OR } -1=-1$ ; $0 \text{ OR } -1=-1$ ; $0 \text{ OR } 0 = 0$ .                                                                                                                      |
| <b>OUT P,N</b>                    | Sends the value N to port P. Both arguments must be in the range 0 to 255; useful with an expansion interface.                                                                                                     |
| <b>PEEK(address)</b>              | Returns the value stored in the specified address.                                                                                                                                                                 |
| <b>POINT(X,Y)</b>                 | A graphic function to test whether the graphic block at the horizontal position X and vertical position Y is turned on or off. If it is on, the function returns a -1, and if it is not turned on, it returns a 0. |
| <b>POKE address, N</b>            | Stores the value N into the memory location of the specified address. N must be between 0 and 255.                                                                                                                 |

|                                 |                                                                                                                                                                                                                                                                                                                    |
|---------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>POS(X)</b>                   | Determines the current cursor position: a number from 0 to 63. X is any (dummy) argument.                                                                                                                                                                                                                          |
| <b>PRINT "MESSAGE",<br/>A,B</b> | Displays the message (if any) and the values of A and B in successive zones of the screen.                                                                                                                                                                                                                         |
| <b>PRINT "MESSAGE";<br/>A;B</b> | Displays the message (if any) and the values of A and B with few spaces between them.                                                                                                                                                                                                                              |
| <b>PRINT USING</b>              | Specifies a format for displaying numeric and string values. Useful in report generation.                                                                                                                                                                                                                          |
| <b>PRINT @ N,A,B</b>            | Displays the values of A and B starting at position N. The screen is divided into a grid of 1024 positions numbered 0 to 1023.                                                                                                                                                                                     |
| <b>PRINT #-1, A,B</b>           | Prints the values of A and B onto a cassette tape. The list to be printed cannot exceed 255 characters. These data can subsequently be read into the computer with the statement <b>INPUT #-1,A,B</b> .                                                                                                            |
| <i>Program</i>                  | A set of instructions in BASIC to perform a specific task.                                                                                                                                                                                                                                                         |
| <i>Programming mode</i>         | A form of the command mode; each statement must have a line number.                                                                                                                                                                                                                                                |
| <i>Radians</i>                  | A measure of angles; 1 radian = 57.296 degrees.                                                                                                                                                                                                                                                                    |
| <i>RAM</i>                      | Random Access Memory; memory that is available for storing programs and data.                                                                                                                                                                                                                                      |
| <b>RANDOM</b>                   | Reseeds the random-number generator and ensures that the computer generates a new sequence of random numbers.                                                                                                                                                                                                      |
| <i>Random access</i>            | Files may be accessed directly as, for example, on a mini disk. This is in contrast to sequential access of files on tape.                                                                                                                                                                                         |
| <i>Random number</i>            | A number whose value cannot be predicted from the numbers that precede it.                                                                                                                                                                                                                                         |
| <b>READ A,B</b>                 | Reads values for A and B from a <b>DATA</b> statement.                                                                                                                                                                                                                                                             |
| <b>REDO</b>                     | Computer's response to entry of inappropriate numeric or string data for <b>INPUT</b> .                                                                                                                                                                                                                            |
| <i>Relational operators</i>     | Compare numeric and string variables. Return logical results; -1 for true, and 0 for false. In numeric expressions, the numbers are compared; in string expressions, the ASCII values are compared: less than (<), greater than (>), equal (=), less than or equal (<=), greater than or equal (>=), unequal (<>). |
| <b>REM</b>                      | A remark used to clarify the program; an explanatory comment. The computer ignores the statement.                                                                                                                                                                                                                  |
| <i>Reserved words</i>           | Words reserved for Level II BASIC and DISK BASIC. These may not be used as variable names.                                                                                                                                                                                                                         |
| <b>RESET(X,Y)</b>               | A graphic function to turn off the graphic block at the horizontal position X and vertical position Y. The point (0,0) is at the top left corner of the screen; $0 \leq X < 128$ and $0 \leq Y < 48$ .                                                                                                             |
| <b>RESTORE</b>                  | The next <b>READ</b> statement following <b>RESTORE</b> will read data from the very first <b>DATA</b> statement in the program.                                                                                                                                                                                   |
| <b>RESUME N</b>                 | Terminates an error-trapping routine and causes execution to transfer unconditionally to line N. <b>RESUME</b> will transfer to the line at which the error occurred. <b>RESUME NEXT</b> will transfer to the line following the error.                                                                            |
| <b>RETURN</b>                   | Ends a subroutine and returns execution to statement following the <b>GOSUB</b> .                                                                                                                                                                                                                                  |
| <b>RIGHT\$(string,N)</b>        | Isolates the last N characters of string.                                                                                                                                                                                                                                                                          |
| <b>RND(X)</b>                   | A library function to generate a random number. <b>RND(0)</b> produces a random number between 0 and 1. <b>RND(10.5)</b> produces a random number between 1 and 10.                                                                                                                                                |

|                                  |                                                                                                                                                                                                                                                              |
|----------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>ROM</b>                       | Read Only Memory; memory occupied by system programs; this memory can only be read and not used for storage.                                                                                                                                                 |
| <b>Routine</b>                   | A set of instructions to perform a specific task.                                                                                                                                                                                                            |
| <b>RUN</b>                       | Command to request execution of a program. Entry of execution mode. <b>RUN N</b> requests that execution start at line N. The <b>RUN</b> command also automatically executes a <b>CLEAR</b> command; sets all numeric variables to zero and strings to null. |
| <b>Scientific notation</b>       | Breaks a number into two parts: a number between 1 and 10 and an exponent of 10; for example, 1000000=1E6; exponent characters E and D for single and double precision, respectively.                                                                        |
| <b>Semicolon</b>                 | Causes the items of a list in a <b>PRINT</b> to be displayed close together with few spaces; for example, <b>PRINT A;B;C\$</b> .                                                                                                                             |
| <b>Sequential access</b>         | Files are stored sequentially one after the other (for example, on tape) and are consequently accessed sequentially. To access the last file on tape, it is necessary to go through the entire tape. This is in contrast to random access on disks.          |
| <b>SET (X, Y)</b>                | A graphic function to turn on the graphic block at the horizontal position X and vertical position Y. The point (0, 0) is at the top left corner on the screen; $0 \leq X < 128$ and $0 \leq Y < 48$ .                                                       |
| <b>SGN(X)</b>                    | A library function that examines the argument X and returns a -1 if X is negative, a 0 if X is zero, and a 1 if X is positive.                                                                                                                               |
| <b>SHIFT key</b>                 | Press to type uppercase characters.                                                                                                                                                                                                                          |
| <b>SHIFT @ key</b>               | Stops execution and freezes the display; to resume, press any key.                                                                                                                                                                                           |
| <b>SHIFT → key</b>               | Converts to 32 characters per line display. To convert back to 64 characters, press <b>CLEAR</b> key.                                                                                                                                                        |
| <b>SHIFT ← key</b>               | Returns cursor to start of current line and erases entire line.                                                                                                                                                                                              |
| <b>Single-precision variable</b> | Six significant figures; for example, $A! = 1.23456$ . All variables are in single precision unless otherwise declared.                                                                                                                                      |
| <b>SIN(X)</b>                    | A library function to determine the sine of X. The argument X must be in radians.                                                                                                                                                                            |
| <b>Source program</b>            | A program written in a language that must be translated into machine language before it can be executed.                                                                                                                                                     |
| <b>SQR(X)</b>                    | A library function to determine the square root of X. The argument X cannot be negative.                                                                                                                                                                     |
| <b>Statement</b>                 | A BASIC instruction.                                                                                                                                                                                                                                         |
| <b>STEP N</b>                    | Part of the <b>FOR</b> statement; the <b>STEP N</b> increments the loop's counter by N during each sweep.                                                                                                                                                    |
| <b>STOP</b>                      | Interrupts execution and displays the line at which execution was stopped. The command <b>CONT</b> will resume execution.                                                                                                                                    |
| <b>String</b>                    | A sequence of alphanumeric characters; string variables are tagged with a \$ to distinguish them from numeric variables. Strings are enclosed in quotes; for example, $A\$ = \text{"TRS-80"}$ .                                                              |
| <b>String operators</b>          | Strings may be concatenated (+) or compared using relational operators.                                                                                                                                                                                      |
| <b>STRING\$(N,"C")</b>           | Produces a string consisting of N characters C; for example, <b>STRING \$(3,"+")</b> is <b>+++</b> .                                                                                                                                                         |
| <b>STR\$(N)</b>                  | The numeric expression N is converted to a string; for example, <b>STR\$(1.2)</b> is <b>"1.2"</b> . Performs the inverse of the <b>VAL</b> function.                                                                                                         |
| <b>Subroutine</b>                | A subprogram, a section within a program that may be accessed from the program itself or from other subroutines during execution.                                                                                                                            |
| <b>Subscript</b>                 | The expression in parentheses identifying the specific element of an array; for example, the I in <b>A(I)</b> .                                                                                                                                              |

|                                       |                                                                                                                                                                                                                                                                                                |
|---------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>Subscripted variable</i>           | A list of items; for example, <b>A(I)</b> is a one-dimensional array; <b>A\$(I,J)</b> is a two-dimensional array containing strings; <b>A%(I,J,K,L)</b> is a four-dimensional array of integer values.                                                                                         |
| <b>SYSTEM</b>                         | Command to enter the monitor mode for loading a machine language file from cassette.                                                                                                                                                                                                           |
| <b>TAB(X)</b>                         | A library function; for example, the statement <b>10 PRINT TAB(20)A</b> will display the value of <b>A</b> starting at the twentieth print position. Each line contains 64 print positions numbered 0 to 63. <b>TAB(65)</b> displays on the next line. The argument must be between 0 and 255. |
| <b>TAN(X)</b>                         | A library function to determine the tangent of <b>X</b> . The argument <b>X</b> must be in radians.                                                                                                                                                                                            |
| <i>Trace</i>                          | A technique for debugging programs. The line number of each statement is displayed as it is executed. <b>TRON</b> and <b>TROFF</b> turn the trace on and off, respectively.                                                                                                                    |
| <i>Transfer statements</i>            | BASIC statements that cause execution to deviate from the normal sequence of execution (starting with the lowest line number and executing subsequent lines in ascending order).                                                                                                               |
| <i>Translator</i>                     | A program that converts a high-level language to another high-level language.                                                                                                                                                                                                                  |
| <b>TROFF</b>                          | Command to turn off the Trace.                                                                                                                                                                                                                                                                 |
| <b>TRON</b>                           | Command to request a trace of the program during execution. Useful in debugging; the computer displays the line number of each line as it is executed. The <b>TROFF</b> command terminates the trace. <b>TRON</b> and <b>TROFF</b> may be used within a program.                               |
| <i>Two-dimensional array</i>          | A subscripted variable with two subscripts; for example, <b>A(I,J)</b> .                                                                                                                                                                                                                       |
| <i>Type definition</i>                | A statement that declares variables as integer, string, single, or double precision.                                                                                                                                                                                                           |
| <i>Unconditional transfer</i>         | Deviate from the usual sequential execution and branch to a specific line in the program; for example, <b>10 GOTO 50</b> .                                                                                                                                                                     |
| <b>USR(X)</b>                         | Transfers to a machine language subroutine.                                                                                                                                                                                                                                                    |
| <b>VAL(string)</b>                    | Converts the digits in string to a number; for example, <b>VAL("1984")</b> is the numeric constant 1984. Performs the inverse of the <b>STR\$</b> function.                                                                                                                                    |
| <i>Variable</i>                       | A quantity represented by a symbol that can assume various values.                                                                                                                                                                                                                             |
| <i>Variable declaration character</i> | Identifies the variable type; a character is appended to the variable name: <b>!</b> , single precision; <b>#</b> , double precision; <b>%</b> , integer; <b>\$</b> , string. Without a declaration, variables are assumed to be single precision.                                             |
| <i>Variable name</i>                  | Identifies the variable; must begin with a letter; may be followed by a digit or a second letter. Additional letters or digits are ignored by the computer. Valid names <b>T</b> , <b>TR</b> , <b>TRS</b> . Variables <b>TR</b> and <b>TRS</b> are the same.                                   |
| <i>Variable types</i>                 | Four different types: single precision, double precision, integer, and string.                                                                                                                                                                                                                 |
| <b>VARPTR(V)</b>                      | Produces an address to help locate where the variable name <b>V</b> and its value are stored.                                                                                                                                                                                                  |
| <i>Zero subscripted element</i>       | First element in an array, <b>A(0)</b> .                                                                                                                                                                                                                                                       |
| <i>Zone</i>                           | The screen is divided into 4 zones, each containing 16 print positions.                                                                                                                                                                                                                        |

## SOLUTIONS TO EVEN-NUMBERED EXERCISES

### CHAPTER 2 EXERCISES 1

CH. 2, EX. 1, PROBLEM 2  
PRINT 15  
15

CH. 2, EX. 1, PROBLEM 4  
PRINT "LEONID BREZHNEV"  
LEONID BREZHNEV  
PRINT "SOVIET UNION"  
SOVIET UNION

CH. 2, EX. 1, PROBLEM 6-A  
ALPHABET\$="ABCDEFGHIJKLMNOPQRSTUVWXYZ"  
PRINT ALPHABET\$  
ABCDEFGHIJKLMNOPQRSTUVWXYZ

CH. 2, EX. 1, PROBLEM 6-B  
N\$="0123456789"  
PRINT N\$  
0123456789

\*\*\*\*\*

### CHAPTER 2 EXERCISES 2

CH. 2, EX. 2, PROBLEM 2  
SP!=4869684.5\*8  
PRINT SP!  
3.89575E+07  
DP#=4869684.5\*8  
PRINT DP#  
38957476

CH. 2, EX. 2, PROBLEM 4  
A) A+B-C  
B) C\*(A-Y)  
C) A\*B-2  
D) A\*B\*C\*A  
E) C-B/3-D  
F) (A\*B-C\*(A-B))/(3-A)-A\*C

CH. 2, EX. 2, PROBLEM 6  
AB=5  
AC=6  
PRINT AB+AC  
11  
PRINT AB-AC  
-1  
PRINT AB\*AC  
30  
PRINT AB/AC  
933333

CH. 2, EX. 2, PROBLEM 8  
A) ?/0 ERROR  
B) 0  
C) 1  
D) 0  
E) ?/0 ERROR

CH. 2, EX. 2, PROBLEM 10  
A) PRINT 40\*9.50  
B) PRINT 37\*9.50  
C) PRINT 40\*9.50+7\*14.25

CH. 2, EX. 2, PROBLEM 12  
NOTE: S=(T\*60+M)\*60  
PRINT (1\*60+30)\*60  
5400  
PRINT (0\*60+1)\*60  
60

CH. 2, EX. 2, PROBLEM 14  
P=24  
I=.03  
N=1980-1627  
BALANCE=P\*(1+I)<sup>N</sup>  
PRINT BALANCE  
816100

CH. 2, EX. 2, PROBLEM 16  
I=.08/12  
PRINT 30000\*I\*(1+I)<sup>[480/((1+I)<sup>[480-1]</sup></sup>  
208.594  
I=.1/12  
PRINT 30000\*I\*(1+I)<sup>[480/((1+I)<sup>[480-1]</sup></sup>  
254.744

CH. 2, EX. 2, PROBLEM 18  
A) SLOPE=(Y2-Y1)/(X2-X1)  
B) X1=1  
X2=4  
Y1=2  
Y2=8  
DISTANCE=((X2-X1)<sup>[2+(Y2-Y1)<sup>[2]]</sup>)<sup>[.5]</sup>  
PRINT DISTANCE  
6.70821  
SLOPE=(Y2-Y1)/(X2-X1)  
PRINT SLOPE  
2</sup>

\*\*\*\*\*

### CHAPTER 3 EXERCISES 3

10 REM CH. 3, EX. 3, PROBLEM 2  
20 PRINT "MY COMPUTER LIKES ME"  
30 C=100 'STORE 100 IN C  
40 PRINT C; "YEARS AGO"  
50 B=10 'STORE 10 IN B  
60 A=1 'STORE 1 IN A  
70 PRINT A+B+C; "YEARS AGO"  
75 'NOTE: SEMICOLONS ARE OPTIONAL  
80 PRINT A; "PLUS" B "EQUALS" A+B  
90 PRINT "HE HAS "; -B; "DOLLARS"

10 REM CH. 3, EX. 3, PROBLEM 4  
15 I=.08 'SET I=INTEREST RATE  
20 B=10000 'B IS BALANCE  
30 PRINT "INTEREST PAID ON"; B  
40 PRINT "FOR A DAY IS"; I/365\*B  
50 PRINT "FOR A WEEK IS"; I/52\*B  
55 'NOTE: SEMICOLON OPTIONAL  
60 PRINT "FOR 3 MONTHS IS"; I/4\*B  
70 PRINT "FOR 6 MONTHS IS"; I/2\*B  
80 PRINT "FOR A YEAR IS"; I\*B



```
10 REM CH. 3, EX. 3, PROBLEM 6-A
20 PRINT "IN FRENCH RED IS ROUGE"
30 PRINT "IN GERMAN IT IS ROT"
40 'NOTE: TRAILING QUOTES OPTIONAL

10 REM CH. 3, EX. 3, PROBLEM 6-B
20 X=2
30 Y=3
40 PRINT "THE SUM OF X AND Y IS:"X+Y
50 PRINT "THE PRODUCT OF X AND Y IS:"X*Y
```

```
10 REM CH. 3, EX. 3, PROBLEM 8
20 A=10
25 B=20
27 PRINT "QUOTIENT OF";A;"TO";B;"IS";A/B
30 'NOTE: B CANNOT BE ZERO
```

```
10 REM CH. 3, EX. 3, PROBLEM 10-A
20 S=10
30 PRINT "THE VOLUME IS"
40 PRINTS*S*S
```

```
10 REM CH. 3, EX. 3, PROBLEM 10-B
20 S=10
30 PRINT "THE VOLUME IS";S*S*S
```

```
10 REM CH. 3, EX. 3, PROBLEM 10-C
20 S=10
30 PRINT "THE SIDE IS";S
```

```
10 REM CH. 3, EX. 3, PROBLEM 10-D
20 S=10
30 PRINT "THE SIDE IS";S
40 PRINT "THE VOLUME IS";S*S*S
50 REM
60 REMARKS: REPLACE LINE 20 BY S=20
```

```
10 REM CH. 3, EX. 3, PROBLEM 12
20 PRINT " X"
30 PRINT " XXX"
40 PRINT " XXXXX"
50 PRINT "XXXXXXXXX"
60 PRINT " XXX"
70 PRINT " XXX"
```

\*\*\*\*\*

#### CHAPTER 3 EXERCISES 4

```
CH. 3, EX. 4, PROBLEM 2
A) REENTER LINE 30 WITH THE
 CORRECT VERSION.
B) TYPE 'EDIT 30' THEN PRESS
 'X' DELETE CHARACTERS UNTIL
 REACH 'A' OF PRODUCT, NEXT
 TYPE 'UCT OF A AND B IS ";A*B'.
C) TYPE 'EDIT 30' TO ENTER EDIT MODE
 PRESS THE SPACE BAR UNTIL THE 'D'
 OF PRODUCT SHOWS, TYPE 'C' TO CHANGE
 THE NEXT CHARACTER; NOW
 TYPE 'U' AND THEN 'ENTER'.
```

```
CH. 3, EX. 4, PROBLEM 4
10 PRINT 'USE D OPTION
15 INPUT A 'USE I OPTION
20 PRINT "PHYSICS IS FUN" 'USE X OPTION
25 PRINT A;B;C 'USE C OPTION
30 INPUT "YOUR AGE";AGE 'USE X OPTION
40 C=5*(F-32)/5 'USE C OPTION
45 PRINT "DON'T TREAD" 'USE K OPTION
50 PRINT "ONE FOR THE M$NEY" 'USE S&C OPT
55 X=5 'USE D OPTION
99 END 'RETYPE LINE 35 AS 99, TYPE 35
```

```
10 REM CH. 3, EX. 4, PROBLEM 8
20 'MONTHLY MORTGAGE PAYMENTS
30 INPUT "AMOUNT & YRS OF MORTGAGE";M;Y
40 INPUT "ENTER INTEREST RATE IN %";I
50 'CONVERT MORT. DURATION FROM Y TO M
60 N=Y*12
65 'CALCULATE MONTHLY INTEREST
70 I=I*.01/12
75 'CALCULATE MONTHLY PAYMENTS
80 P=M*I*(1+I)^N/((1+I)^N-1)
90 PRINT "MONTHLY PAYMENT IS $";P
```

```
10 REM CH. 3, EX. 4, PROBLEM 10-A
20 INPUT "FIRST TEMP";F
30 INPUT "SECOND TEMP";S
40 INPUT "THIRD TEMP";T
50 PRINT "AVERAGE TEMP IS"(F+S+T)/3
```

```
10 REM CH. 3, EX. 4, PROBLEM 10-B
20 INPUT "ENTER 3 TEMPS";F,S,T
30 PRINT "AVERAGE TEMP IS"(F+S+T)/3
```

```
10 REM CH. 3, EX. 4, PROBLEM 12
20 PRINT "HELLO"
30 INPUT "HI WHAT IS YOUR NAME";N$
40 PRINT "HELLO ";N$
50 PRINT "MY NAME IS NEUTER COMPUTER"
```

```
10 REM CH. 3, EX. 4, PROBLEM 14
20 PRINT "PLEASE ANSWER THE FOLLOWING
 QUESTION"
40 INPUT "DO YOU PREFER BOYS OR
 GIRLS";C$
50 PRINT "HEY, I TOO PREFER "C$"
 THAT'S GREAT!"
```

```
10 REM CH. 3, EX. 4, PROBLEM 16
20 INPUT "DEPTH IN FEET";H
30 P=0.0295*H
40 PRINT "AT A DEPTH OF";H;"FEET THE ";
45 'NOTE THE SEMICOLON AT THE END OF
46 ' LINE 40 SUPPRESSES THE LINE
47 ' FEED -- SEE CHAPTER 5
50 PRINT "PRESSURE IS";P;"ATMOSPHERES."
```

\*\*\*\*\*

#### CHAPTER 4 EXERCISES 6

```
10 REM CH. 4, EX. 6, PROBLEM 4-A
20 INPUT "ENTER 2 NUMBERS";N1,N2
25 IF N1+N2<=0 THEN 40
30 PRINT "THE SUM OF "N1"+"N2"=";N1+N2
40 END
```

```
10 REM CH. 4, EX. 6, PROBLEM 4-B
20 INPUT "TWO NUMBERS"; N1, N2
30 IF N1+N2<=N1*N2 THEN 40
35 PRINT "SUM IS GREATER": END
40 IF N1+N2=N1*N2 THEN 50
45 PRINT "PRODUCT IS GREATER": END
50 PRINT "PRODUCT EQUALS SUM"
```

```
10 REM CH. 4, EX. 6, PROBLEM 4-C
20 INPUT "THE CAPITAL OF FRANCE"; CAP$
30 IF CAP$="PARIS" PRINT "YOU ARE OK": END
40 PRINT "NOPE, SEE YOU IN PARIS."
```

```
10 REM CH. 4, EX. 6, PROBLEM 6
15 INPUT "ENTER 4 NUMBERS"; A, B, C, D
20 IF A>B AND A>C AND A>D PRINT A: GOTO 60
30 IF B>A AND B>C AND B>D PRINT B: GOTO 60
40 IF C>A AND C>B AND C>D PRINT C: GOTO 60
50 PRINT D
60 PRINT "IS LARGEST": END
```

```
10 REM CH. 4, EX. 6, PROBLEM 8
15 INPUT "SALES"; S
20 IF S<=1000 THEN PAY=100: GOTO 50
30 IF S<=2000 THEN PAY=.15*S: GOTO 50
40 PAY=250+.07*S
50 PRINT "PAY IS"; PAY
```

```
10 REM CH. 4, EX. 6, PROBLEM 10
15 INPUT "AGE"; A
17 IF A<=0 PRINT "INVALID AGE": GOTO 15
18 IF A>120 PRINT "INVALID AGE": GOTO 15
20 IF A>=18 THEN 30
25 PRINT "PERSON IS A MINOR": END
30 IF A>=65 THEN 50
40 PRINT "PERSON IS NEITHER A MINOR"
41 PRINT "NOR A SENIOR CITIZEN": END
50 PRINT "PERSON IS A SENIOR CITIZEN"
```

```
10 REM CH. 4, EX. 6, PROBLEM 12
15 ON ERROR GOTO 50
18 INPUT "A, B, C"; A, B, C
20 X=(-B+(B*B-4*A*C)^(.5))/(2*A)
30 PRINT "REAL ROOT IS"; X
40 END
50 PRINT "IMAGINARY ROOT"
60 RESUME 40
```

```
10 REM CH. 4, EX. 6, PROBLEM 14
20 INPUT "NUMBER"; N%
40 TEST%=N%/2
60 IF N%=2*TEST% PRINT "EVEN": END
90 PRINT "ODD"
```

\*\*\*\*\*

## CHAPTER 5 EXERCISES 7

```
CH. 5, EX. 7, PROBLEM 4 A-E
A) 10 PRINT N+1
20 IF N<19 THEN N=N+1: GOTO 10
OR 10 FOR N=1 TO 20
15 PRINT N: NEXT N
```

```
B) 10 PRINT N+1: IF N>=18 THEN END
15 N=N+2: GOTO 10
OR 10 FOR N=1 TO 20 STEP 2
15 PRINT N: NEXT N
```

```
C) 10 N=7
20 PRINT N: IF N>=14 THEN END
25 N=N+.5: GOTO 20
OR 10 FOR N=7 TO 14 STEP .5
15 PRINT N: NEXT N
```

```
D) 10 N=3
20 PRINT 1/N
25 IF N=10 THEN END
30 N=N+1: GOTO 20
OR 10 FOR N=3 TO 10
15 PRINT 1/N: NEXT N
```

```
E) 10 N=10
20 PRINT N: N=N-3
25 IF N>=-10 THEN 20
OR 10 FOR N=10 TO -10 STEP -3
15 PRINT N: NEXT N
```

```
CH. 5, EX. 7, PROBLEM 4-F
10 N=2: P=1
20 P=P*N: N=N+2: IF N<=10 THEN 20
30 PRINT "PRODUCT"; P
40 REM USING A FOR-NEXT LOOP
50 P=1
55 FOR N=2 TO 10 STEP 2
60 P=P*N: NEXT N
70 PRINT "PRODUCT"; P
```

```
CH. 5, EX. 7, PROBLEM 4-G
10 INPUT A, B: P=1
13 IF A<B THEN 19
14 REM SWAP A & B
15 C=A: A=B: B=C
19 N=A
20 P=P*N: N=N+1: IF N<=B THEN 20
30 PRINT "PRODUCT"; P
40 REM USING A FOR-NEXT LOOP
45 P=1
50 FOR N=A TO B: P=P*N: NEXT N
60 PRINT "PRODUCT"; P
```

```
CH. 5, EX. 7, PROBLEM 6
10 ' RULE OF 72
20 INPUT "DEPOSIT & INTEREST RATE"; DE, IN
40 BA=DE
60 FOR N=1 TO 10000
70 BA=BA+BA*IN/100
80 PRINT "BALANCE FOR YEAR"; N: "IS"; BA
82 IF BAL>=DE*2 THEN 90
85 NEXT N
90 PRINT "TIME TO DOUBLE IS"; N: "YEARS"
95 PRINT "FINAL BALANCE IS $"; BA
```

```
10 REM CH. 5, EX. 7, PROBLEM 8
15 PRINT "SEC X Y"
18 X=80*T
20 Y=4+70*T-16.1*T*T
40 IF X>355 THEN 70
45 PRINT T: " "; X: " "; Y
50 T=T+.5: GOTO 18
70 PRINT "OUT OF THE BALL PARK"
```

```

10 REM CH. 5, EX. 7, PROBLEM 10
15 CLS: FLAG=0: PROF=-51
18 T=T+1
20 P=T*T*T-5*T*T+10*T-51
22 PROF=PROF+P
24 PRINT T,P,PROF
25 IF T=8 THEN 60
30 IF P>0 AND FLAG=0 THEN 40
35 GOTO 18
40 FLAG=1
45 PRINT "PROFIT IS 1ST POS. IN YEAR":T
50 GOTO 18
60 PRINT "CUMULATIVE PROFIT":PROF*1000

```

```

10 REM CH. 5, EX. 7, PROBLEM 12
15 PRINT "I WILL GUESS YOUR NUMBER AND"
18 PRINT "YOU TELL ME IF MY GUESS IS"
19 PRINT "HIGH, CORRECT, OR LOW"
20 L=1
23 H=100
30 GX=(L+H)/2
40 PRINT "GUESS IS":GX: "H, C, OR L":
45 INPUT Q#
50 IF Q#="H" LET H=GX: GOTO 30
60 IF Q#="L" LET L=GX: GOTO 30
70 IF Q#<>"C" PRINT Q#:"?":GOTO40
80 PRINT "AH I WAS RIGHT"

```

```

10 REM CH. 5, EX. 7, PROBLEM 14
15 D=1
20 T=T+1/D: IF T=4 THEN 40
30 D=D+1: GOTO 20
40 PRINT D: "TERMS SUM UP TO 4"

```

```

10 REM CH. 5, EX. 7, PROBLEM 16
20 INPUT "# OF TERMS":T
30 F=1: D=1: X=0: K=1
40 X=X+(1/D)*F: D=D+2: F=-F: K=K+1
50 IF K<=T THEN 40
70 PRINT "FOR":T: "TERMS, SUM IS":4*X

```

```

10 REM CH. 5, EX. 7, PROBLEM 18
14 P=1000:F=100000
15 PRINT "YEAR "; "POPULATION "; "FOOD"
20 Y=Y+10: P=2*P: F=F+4000
30 PRINT Y: " "; P: " "; F
40 IF F<P PRINT "FOOD OUT IN YEAR":Y:END
50 GOTO 20

```

```

10 REM CH. 5, EX. 7, PROBLEM 20
15 B=D
20 INPUT "MONTHLY DEPOSIT":D
25 FOR X=1 TO 12
30 B=D+B*B*.06/12
35 NEXT X
40 PRINT "AFTER A YEAR YOU HAVE $":B

```

```

10 REM CH. 5, EX. 7, PROBLEM 22
12 PRINT "PLEASE BE PATIENT"
15 FOR N=1 TO 999
18 REM GIVEN RELATION BETWEEN X & N
20 X#=-0.5+(-0.25+0.5*N+0.5*N*N)/0.5
25 REM X MUST BE AN INTEGER
30 IF X#<100 THEN 45
35 REM X ROUNDED TO 3 PLACES

```

```

36 / MUST BE AN INTEGER
40 X%=X#: X1=X#-X%: X%=X1*1000
43 IF X%=0 THEN 50
45 NEXT N
50 X=X#: PRINT "N=":N: "X=":X
55 END
60 REM NOTE, ANSWER N=696 & X=492

```

\*\*\*\*\*

# CHAPTER 5 EXERCISES 8

```

10 REM CH. 5, EX. 8, PROBLEM 4-A
20 FOR Y=1 TO 6: PRINT Y: NEXT Y

```

```

10 REM CH. 5, EX. 8, PROBLEM 4-B
20 FOR Y=1 TO 6: PRINT Y: NEXT Y

```

```

10 REM CH. 5, EX. 8, PROBLEM 4-C
20 FOR X=1 TO 3: PRINT X: NEXT X
25 PRINT
30 FOR X= 4 TO 6: PRINT X: NEXT X

```

CH. 5, EX. 8, PROBLEM 6

```

NEW
PRINT MEM
15572
DIM A(10)
PRINT MEM
15520

```

```

10 REM CH. 5, EX. 8, PROBLEM 8
20 DIM M(30)
30 FOR X=1 TO 30: M(X)=X: NEXT X

```

```

10 REM CH. 5, EX. 8, PROBLEM 10
15 FOR X=1 TO 10
20 PRINT "ENTER NUMBER":X
25 INPUT A(X)
30 NEXT X
35 FOR X=10 TO 1 STEP -1
40 PRINT A(X):
45 NEXT X

```

```

10 REM CH. 5, EX. 8, PROBLEM 12
20 FOR X=1 TO 5
30 INPUT "NAME":N$(X)
35 INPUT "AGE":A(X)
40 NEXT
50 FOR X=1 TO 5
55 IF A(X)<=65 THEN 65
60 PRINT N$(X): " AGE":A(X)
65 NEXT X

```

```

10 REM CH. 5, EX. 8, PROBLEM 14-A
15 DIM N$(50)
20 FOR L=1 TO 5
30 INPUT "NAME":N$(L)
40 NEXT L
50 FOR L=1 TO 4: FOR I=L+1 TO 5
60 IF N$(L)<N$(I) THEN 70
65 S$=N$(L): N$(L)=N$(I): N$(I)=S$
70 NEXT I,L
75 PRINT "NAME CLOSEST TO 'A' ":N$(1)
80 PRINT "NAME CLOSEST TO 'Z' ":N$(5)

```

```

CH. 5, EX. 8, PROBLEM 16
10 FOR C=1 TO 6
20 PRINT "BALANCE FOR MONTH";C;" WAS"
25 INPUT T(C)
30 NEXT C
35 T=710
40 FOR C=1 TO 6: D(C)=T(C)-T:T=T(C)
50 PRINT "NET DEPOSIT MONTH";C;" WAS";D(C)
60 NEXT C
70 END

```

```

10 REM CH. 5, EX. 8, PROBLEM 18
15 M=3: N=2
20 FOR Y=1 TO N
25 FOR X=1 TO M
30 PRINT "INPUT A("X","Y")"
35 INPUT A(X,Y)
40 PRINT "INPUT B("X","Y")"
45 INPUT B(X,Y)
50 NEXT X,Y
55 FOR X=1 TO M: FOR Y=1 TO N
60 IF A(X,Y) > B(X,Y) THEN 70
65 C(X,Y)=B(X,Y): GOTO 75
70 C(X,Y)=A(X,Y)
75 PRINT C(X,Y)
80 NEXT Y
85 PRINT: NEXT X

```

```

10 REM CH. 5, EX. 8, PROBLEM 20
15 M=3: N=4: DIM M(3,4)
20 FOR X=1 TO M
30 FOR Y=1 TO N
40 PRINT "ENTER ";X;" ";Y;
45 INPUT M(X,Y)
50 NEXT Y,X
60 FOR X=1 TO M
65 FOR Y=1 TO N
70 IF M(X,Y)<=100 THEN 85
75 NEXT Y
80 PRINT "ROW ";X
85 NEXT X
90 FOR Y=1 TO N
92 FOR X=1 TO M
94 IF M(X,Y)<=100 THEN 99
96 NEXT X
98 PRINT "COLUMN ";Y
99 NEXT Y

```

```

10 REM CH. 5, EX. 8, PROBLEM 22
12 DIM A(20), B(21)
15 INPUT "THE NUMBER TO BE MERGED";X
20 PRINT "ENTER THE SORTED ARRAY"
23 FOR K=1 TO 10
25 PRINT "ELEMENT NO. ";K
28 INPUT A(K): NEXT
30 FOR K=1 TO 10
40 IF A(K)>X THEN 60
50 B(K)=A(K)
55 GOTO 80
60 B(K)=X
65 B(K+1)=A(K)
70 GOTO 90
80 NEXT K
82 B(K)=X
84 GOTO 99
90 FOR L=K+2 TO 11
92 B(L)=A(L-1)
94 NEXT L
99 FOR L=1 TO 11:PRINT B(L): NEXT

```

```

10 REM CH. 5, EX. 8, PROBLEM 24A
13 DIM N(20): CLS
15 PRINT "ENTER THE FIRST ARRAY"
16 PRINT "SORTED IN DESCENDING ORDER"
20 FOR X=1 TO 5
25 PRINT "NUMBER";X
26 INPUT N(X)
30 NEXT X
35 PRINT "ENTER THE SECOND ARRAY"
36 PRINT "SORTED IN DESCENDING ORDER"
40 FOR X=6 TO 10
45 PRINT "NUMBER";X-5
46 INPUT N(X)
50 NEXT X
55 FOR J=1 TO 9
60 FOR R=J+1 TO 10
65 IF N(J)>N(R) THEN 75
70 S=N(R): N(R)=N(J): N(J)=S
75 NEXT R,J
80 FOR X=1 TO 10:PRINT N(X): NEXT

```

```

10 REM CH. 5, EX. 8, PROBLEM 24B
13 DIM N$(20): CLS
15 PRINT "ENTER THE FIRST STRING"
16 PRINT "SORTED IN DESCENDING ORDER"
20 FOR X=1 TO 5
25 PRINT "ELEMENT";X
26 INPUT N$(X)
30 NEXT X
35 PRINT "ENTER THE SECOND ARRAY"
36 PRINT "SORTED IN DESCENDING ORDER"
40 FOR X=6 TO 10
45 PRINT "ELEMENT";X-5
46 INPUT N$(X)
50 NEXT X
55 FOR J=1 TO 9
60 FOR R=J+1 TO 10
65 IF N$(J)>N$(R) THEN 75
70 S=N$(R): N$(R)=N$(J): N$(J)=S
75 NEXT R,J
80 FOR X=1 TO 10:PRINT N$(X); " ";:NEXT

```

```

10 REM CH. 5, EX. 8, PROBLEM 26
12 PRINT "BE PATIENT"
15 MIN=1E20
17 A=1: D=2: G=3
20 FOR B=4 TO 9
25 FOR C=4 TO 9
30 IF B=C THEN 97
35 FOR E=4 TO 9
40 IF E=B OR E=C THEN 95
45 FOR F=4 TO 9
50 IF F=E OR F=C OR F=B THEN 93
55 FOR H=4 TO 9
60 IF H=B OR H=C OR H=E OR H=F THEN 91
65 FOR I=4 TO 9
70 IF I=H OR I=F OR I=E OR I=C OR I=B THEN 89
75 N1=100+10*B+C
76 N2=200+10*E+F
77 N3=300+10*H+I
80 P=N1*N2*N3
85 IF MIN<P THEN 89
86 S1=N1: S2=N2: S3=N3
87 MIN=P
89 NEXT I
91 NEXT H
93 NEXT F
95 NEXT E
97 NEXT C
98 NEXT B
99 PRINT S1; S2; S3

```

CHAPTER 6  
EXERCISES 9

```
10 REM CH. 6, EX. 9, PROBLEM 2
20 DATA 1, 2, 3, 10, 20, 30
```

```
10 REM CH. 6, EX. 9, PROBLEM 6
15 READ L: ZZ=1: Z=1
17 REM L IS ARRAY LENGTH
20 FOR X=J TO L: READ Y
25 IF Y=0 THEN 30
27 N(Z)=Y: Z=Z+1: GOTO 40
30 P(ZZ)=Y: ZZ=ZZ+1
40 NEXT
45 PRINT "ARRAY P"
50 FOR X=1 TO ZZ-1
55 PRINT P(X): NEXT X
58 PRINT: PRINT "ARRAY N"
60 FOR X=1 TO Z-1
65 PRINT N(X): NEXT X
70 DATA 5, -1, 2, 3, -5, 0
```

```
10 REM CH. 6, EX. 9, PROBLEM 8
12 PRINT "NO. "; TAB(12); "SQUARE"; TAB(24);
13 PRINT "SQ. ROOT"; TAB(36); "CUBE";
14 PRINT TAB(48); "CUBE ROOT"
15 FOR X=1 TO 10
20 PRINT X; TAB(12); X*X; TAB(24);
25 PRINT X(.5); TAB(36); X(.3)
26 PRINT TAB(48); X(.1/3)
30 NEXT X
```

```
10 REM CH. 6, EX. 9, PROBLEM 10
11 REM SALES REPORT
12 FOR I=1 TO 4
15 READ N$(I), ASALES(I), BSALES(I)
18 DATA ERIC, 6, 8, RON, 9, 5, JEFF, 4, 12
19 DATA FRED, 20, 2
21 ACOM(I)=125 * ASALES(I)
24 CHECK=BSALES(I)-10
27 IF CHECK>0 THEN 36
30 BCOM(I)=95*BSALES(I)
33 GOTO 37
36 BCOM(I)=950 + 145*CHECK
37 TCOM(I)=ACOM(I)+BCOM(I)
38 TSALES(I)=ASALES(I)+BSALES(I)
39 NEXT I
42 REM PREPARE REPORT
43 CLS
45 PRINT "SUPER AGENCY SALES REPORT"
46 PRINT
50 PRINT TAB(12); N$(1); TAB(24); N$(2);
51 PRINT TAB(36); N$(3); TAB(48); N$(4)
52 PRINT "A SALES"; TAB(12); AS(1);
53 PRINT TAB(24); AS(2);
54 PRINT TAB(36); AS(3); TAB(48); AS(4)
56 PRINT "COMMISSIONS"; TAB(12); AC(1);
57 PRINT TAB(24); AC(2); TAB(36); AC(3);
58 PRINT TAB(48); AC(4)
59 PRINT "B SALES"; TAB(12); BS(1); TAB(24);
60 PRINT BS(2); TAB(36); BS(3); TAB(48); BS(4)
61 PRINT "COMMISSIONS"; TAB(12); BC(1);
62 PRINT TAB(24); BC(2); TAB(36); BC(3);
63 PRINT TAB(48); BC(4)
64 PRINT
65 PRINT "SALESMAN", "SALES", "COMMISSION"
66 FOR I=1 TO 4
```

```
67 PRINT N$(I), TSALES(I), TCOM(I)
68 NEXT I
70 END
```

```
10 REM CH. 6, EX. 9, PROBLEM 12
15 DIM H(1000), M(100)
20 PRINT "BUILD TAPE, ENTER 555 TO STOP"
21 PRINT "PLACE REWOUND TAPE IN DRIVE"
22 PRINT "PRESS 'RECORD' & 'PLAY'"
24 PRINT "ENTER INTEGERS FROM KEYBRD."
25 PRINT "-250 < INTERGER < 250"
30 INPUT "INTEGER"; N%
40 PRINT #1, N%
50 IF N%<> 555 THEN 30
61 PRINT "REWIND TAPE, PRESS 'PLAY'"
62 PRINT "FOR DUPLICATE DATA CHECK."
65 INPUT "KEY ENTER WHEN READY"; Q$
70 INPUT #1, N%
80 IF H(N%+250)=1 THEN 70
90 H(N%+250)=1: W=W+1: M(W)=N%
91 IF N%<> 555 THEN 70
92 PRINT "CASSETTE #2 IN DRIVE"
93 PRINT "REWIND & PRESS 'RECORD'"
94 INPUT "KEY ENTER WHEN DONE"; Q$
95 FOR X=1 TO W: PRINT #1, M(X)
96 NEXT X
97 REM PLACE END OF FILE
98 IDENTIFIER AT END OF TAPE
99 PRINT #1, 555
```

```
10 REM CH. 6, EX. 9, PROBLEM 14
11 DIM N$(99), A$(99), C$(99)
12 DIM S$(99), Z$(99)
13 CLEAR 2000 : CLEAR STRING SPACE
14 INPUT "IS THIS A NEW FILE Y/N"; Q$
15 IF Q$="N" THEN 90
16 IF Q$<>"Y" THEN 14
17 X=1 : X IS RECORD NUMBER
21 PRINT "ADD, DELETE, STOP ";
22 INPUT "A, D, S"; Q$
23 IF Q$="S" THEN 47
24 IF Q$="D" THEN 64
25 IF Q$<>"A" PRINT "?"; GOTO 21
35 INPUT "NAME (STOP='END')"; N$(X)
36 IF N$(X)="END" LET X=X+1: GOTO 21
37 INPUT "ADDRESS"; A$(X)
38 INPUT "CITY"; C$(X)
40 INPUT "STATE"; S$(X)
42 INPUT "ZIP CODE"; Z$(X)
45 X=X+1: GOTO 35
46 GOTO 21
47 PRINT "REWIND & *RECORD*"
48 INPUT "KEY ENTER WHEN READY"; Q$
54 REM %WRITE ALL RECORDS%
56 FOR Y=1 TO X-1
58 PRINT #1, N$(Y), A$(Y), C$(Y), S$(Y), Z$(Y)
59 PRINT N$(Y)
60 NEXT Y
62 PRINT #1, "END", Q$, Q$, Q$, Q$
63 END
64 REM SCAN RECORDS FOR NAME
65 INPUT "NAME TO FIND"; N1$
66 REM FIND NAME ROUTINE
67 FOR Y=1 TO X-2
68 IF N1$=N$(Y) THEN 74
70 NEXT Y
71 PRINT N1$; " NOT FOUND"
72 GOTO 21
74 REM DELETE RECORD ROUTINE
```

```

80 FOR Z=Y TO X
81 REM DISPLACE DELETED RECORD
82 N$(Z)=N$(Z+1)
83 A$(Z)=A$(Z+1)
84 C$(Z)=C$(Z+1)
85 S$(Z)=S$(Z+1)
86 Z$(Z)=Z$(Z+1)
87 NEXT Z: X=X-1
88 REM DONE
89 GOTO 21
90 PRINT "REWIND TAPE, PRESS PLAY"
91 REM FIRST RECORD
92 X=1
93 INPUT "PRESS ENTER WHEN READY";Q$
94 INPUT #-1, N$(X), A$(X), C$(X), S$(X), Z$(X)
95 PRINT N$(X)
96 IF N$(X)="END" THEN 21
97 X=X+1. GOTO 94

```

```

10 REM CH. 6, EX. 9, PROBLEM 16
11 INPUT "HOW MANY EMPLOYEES";W
12 DIM E(50), N$(50), YD(50)
15 PRINT "HAVE YOU BUILT ";
16 PRINT "A MASTER FILE?"
17 INPUT "RESPOND WITH Y OR N";Q$
18 IF Q$="Y" THEN 32
19 IF Q$<>"N" GOTO 99
20 PRINT "BUILD MASTER FILE"
21 PRINT "REWIND & RECORD "
22 INPUT "PRESS ENTER WHEN READY";Q$
23 FOR E=1 TO W
24 INPUT "NAME";N$
26 INPUT "YEAR TO DATE EARNINGS";YD
28 PRINT #-1, E, N$, YD
30 NEXT E
32 PRINT "**WEEKLY UPDATE*"
34 PRINT "REWIND & PLAY "
36 INPUT "PRESS ENTER WHEN READY";Q$
38 FOR E=1 TO W
40 INPUT #-1, E(E), N$(E), YD(E)
42 NEXT E
44 FOR E=1 TO W
46 PRINT N$(E), " "; "WEEKS EARNINGS";
48 INPUT WE, YD(E)=YD(E)+WE
50 NEXT E
52 INPUT "WRITE ON TAPE";Q$
54 IF Q$="N" THEN 12
56 IF Q$<>"Y" THEN 99
58 FOR E=1 TO W
60 PRINT #-1, E(E), N$(E), YD(E)
61 PRINT E(E), N$(E), YD(E)
62 NEXT E
64 GOTO 21
99 PRINT "INVALID RESPONSE". GOTO 17

```

\*\*\*\*\*

#### CHAPTER 7 EXERCISE 10

CH. 7, EX. 10, PROBLEM 2  
A) REMAINDER WHEN X IS DIV. BY Y.  
B) TRUNCATES X; X MAYBE POS, NEG, OR 0  
C) DECIMAL FRACTION OF X.  
D) BRANCH TO 10, 20, OR 30 IF X IS NEG,  
ZERO OR POS RESPECTIVELY.

```

10 REM CH. 7, EX. 10, PROBLEM 4
20 LPRINT "A) PRINT 9+RND(11)
30 LPRINT "B) PRINT 1+RND(0)

```

```

10 REM CH. 7, EX. 10, PROBLEM 6
12 RANDOM: DIM B(12)
15 INPUT "THROW DIE HOW OFTEN";N
20 CLS: REM THROW ONE DIE
25 FOR K=1 TO N
30 R=RND(6)
35 A(R)=A(R)+1. NEXT K
40 PRINT "FREQ. OF TOSSING ";
42 PRINT "1 THRU 6 WITH 1 DIE"
45 FOR K=1 TO 6: PRINT A(K);: NEXT K
50 REM THROW 2 DICE
52 PRINT
55 FOR K=1 TO N
60 R=RND(6)+RND(6)
65 B(R)=B(R)+1: NEXT K
70 PRINT "FREQ. OF TOSSING 2 THRU 12";
72 PRINT " WITH 2 DICE"
75 FOR K=2 TO 12
80 PRINT K, B(K). NEXT K

```

```

10 REM CH. 7, EX. 10, PROBLEM 8
12 DIM A(100): RANDOM
15 INPUT "NO. OF RANDOM NOS. ";N
20 B=RND(20): A(B)=1
25 FOR K=2 TO N: R=RND(20)
30 REM FOR PART D
35 IF B=R LET F=F+1
40 A(R)=A(R)+1. B=R. NEXT K
50 REM PART A
55 FOR K=1 TO 20
60 TT=TT+K*A(K).NEXT K
65 PRINT "THE AVERAGE IS"; TT/N
70 REM PART B
75 FOR K=1 TO 20
80 TEN=TEN+A(K). NEXT K
85 PRINT "NO. OF VALUES <=10 "; TEN
90 REM PART C
95 FOR K=1 TO 20 STEP 2
100 ODD=ODD+A(K). NEXT K
105 PRINT "NO. OF ODD INTEGERS"; ODD
110 REM PART D
115 PRINT "NO. OF REPEATS"; F

```

```

10 REM CH. 7, EX. 10, PROBLEM 10
12 CLS: G=0: RANDOM: R=RND(100)
15 PRINT "I HAVE A NO. BETWEEN 1 & 100"
20 G=G+1: PRINT "GUESS NO. ";G
25 INPUT X
30 IF X=R THEN 50
35 IF X<R THEN 45
40 PRINT "TOO HIGH". GOTO 20
45 PRINT "TOO LOW". GOTO 20
50 PRINT "YOU DID IT IN";G; "GUESSES"

```

```

10 REM CH. 7, EX. 10, PROBLEM 12
12 REM HEADS=1, TAILS=2
13 REM FLAG=1 FOR HEADS TOSS
15 FLAG=0: SEQ=0: MAX=0. RANDOM
20 INPUT "HOW MANY TOSSES";N
25 IF RND(2)=2 THEN 35
30 FLAG=1: SEQ=1: GOTO 40
35 FLAG=0
40 FOR K=2 TO N
45 IF RND(2)=2 THEN 60
50 IF FLAG=1 LET SEQ=SEQ+1: GOTO 65
53 IF FLAG=0 LET SEQ=1. FLAG=1
55 GOTO 65
60 FLAG=0: SEQ=0

```

```

65 IF SEQ>MAX LET MAX=SEQ
70 NEXT K
75 PRINT "LONGEST RUN OF HEADS";MAX

```

```

10 REM CH. 7, EX. 10, PROBLEM 14
12 FOR N=100 TO 999
15 D1=INT(N/100)
16 D3=N-10*INT(N/10)
17 D2=(N-100*D1-D3)/10
20 X=D1*3+D2*3+D3*3
25 IF N=X PRINT N
30 NEXT N

```

```

10 REM CH. 7, EX. 10, PROBLEM 16
11 PRINT "ANGLE", "SIN(X)", "SERIES"
12 FOR K=1 TO 7
15 READ ANGLE
20 DATA 10, 30, 60, 90, 120, 150, 180
25 X=.0174533*ANGLE
30 SERIES=0: SIGN=-1: P=-1
35 FOR M=1 TO 5
40 P=P+2: FACT=1: SIGN=-SIGN
45 FOR L=1 TO P
50 FACT=FACT*L: NEXT L
55 SERIES=SERIES+SIGN*X*P/FACT
60 NEXT M
65 PRINT ANGLE, SIN(X), SERIES
70 NEXT K

```

```

10 REM CH. 7, EX. 10, PROBLEM 18
11 CLS
12 FOR ANGLE=0 TO 360 STEP 30
20 Y=SIN(.0174533*ANGLE)
35 ' Y IS MAGNIFIED BY 10 & SHIFTED
37 ' BY 12; FOR Y=1 GRAPH SHOWS Y=22
40 PRINT "-"; TAB(10*Y+12); ANGLE
80 NEXT ANGLE

```

```

10 REM CH. 7, EX. 10, PROBLEM 20
12 PRINT "A", "SIN(2*A)",
13 PRINT "2*SIN(A)*COS(A)"
15 FOR X=0 TO 90 STEP 10
20 A=.0174533*X
25 LEFT=SIN(2*A)
30 RIGHT=2*SIN(A)*COS(A)
35 PRINT X, LEFT, RIGHT
40 NEXT X

```

\*\*\*\*\*

#### CHAPTER 8 EXERCISES 11

```

CH. 8, EX. 11, PROBLEM 8
ADD THE FOLLOWING LINES TO THE PROGRAM
105 INPUT "NO. OF BLINKING DISPLAYS";N
125 FOR X=1 TO N
170 NEXT X: END

```

```

10 REM CH. 8, EX. 11, PROBLEM 10
12 W=0:L=0: CLS: RANDOM
15 PRINT "THIS IS A DICE GAME!!"
20 GOSUB 1000
25 P=D1+D2
30 IF P<>12 THEN 45
35 GOSUB 1300
40 GOTO 105

```

```

45 IF P<>7 THEN 60
50 GOSUB 1200
55 GOTO 105
60 PRINT "YOUR POINT IS";P
65 PRINT "PRESS ENTER TO CONT. ROLLING"
70 INPUT G
75 GOSUB 1000
80 IF D1+D2<>7 THEN 95
85 GOSUB 1300
90 GOTO 105
95 IF D1+D2<>P THEN 60
100 GOSUB 1200
105 INPUT "PLAY AGAIN Y OR N";R$
110 CLS
115 IF R$="Y" THEN 15
120 PRINT: PRINT
125 PRINT "YOU WON";W;
128 PRINT "AND LOST";L; "GAMES. "
130 IF W>L THEN 145
135 PRINT "NEXT TIME BRING MONEY"
140 END
145 PRINT "I DON'T WANT TO PLAY FOR $"
150 END

```

```

1000 REM SUBROUTINE FOR DICE ROLL
1010 D1=INT(RND(6)+1)
1020 D2=INT(RND(6)+1)
1030 PRINT
1040 PRINT"ROLL IS A '";D1;"' & '";D2;"'
1050 PRINT
1060 RETURN
1200 REM WIN ROUTINE
1210 PRINT
1220 PRINT "YOU WIN !!!"
1225 PRINT
1230 W=W+1
1240 RETURN
1300 REM LOSE ROUTINE
1310 PRINT
1320 PRINT "I LOVE A GOOD LOSER!!!"
1325 PRINT
1330 L=L+1
1340 RETURN

```

```

10 REM CH. 8, EX. 11, PROBLEM 12
12 CLS: PRINT "N", "A"
15 READ P, R, T
20 FOR K=1 TO 9
25 READ N(K)
30 DATA 100, .05, 10, 1, 2, 4
35 DATA 8, 16, 32, 64, 128, 365
40 GOSUB 100
45 NEXT K: END
100 REM COMPOUND INTEREST SUBROUTINE
105 A=P*(1+R/N(K))^I (T*N(K))
110 PRINT N(K), A
115 RETURN

```

```

10 REM CH. 8, EX. 11, PROBLEM 14
12 PRINT "INPUT X FOR EXP(X)";
13 PRINT " AND THE NUMBER OF TERMS"
15 INPUT X, N
20 GOSUB 100
25 PRINT N; "TERMS, EXP('";X;"') IS"; E
30 END
100 'SUBROUTINE TO COMPUTE EXP(X)
105 E=1
110 FOR L=1 TO N
115 GOSUB 200
120 E=E+X*E/L/F

```

```

125 NEXT L
130 RETURN
200 REM SUBROUTINE FOR FACTORIAL
205 F=1
210 FOR K=1 TO L
215 F=F*K: NEXT K
220 RETURN

```

\*\*\*\*\*

# CHAPTER 9 EXERCISES 12

```

10 REM CH. 9, EX. 12, PROBLEM 4
20 REM X+W MUST BE <=127
21 ' Y+H MUST BE <=47
30 INPUT "WIDTH & HEIGHT"; W, H
35 INPUT "TOP LEFT CORNER AT X, Y"; A, B
38 CLS
40 Y=B
45 FOR X=A TO A+W: SET (X,Y): NEXT X
50 X=A+W
55 FOR Y=B TO B+H: SET (X,Y): NEXT Y
60 Y=B+H
65 FOR X=A+W TO A STEP -1
66 SET(X,Y): NEXT X
70 X=A
75 FOR Y=B+H TO B STEP -1
76 SET(X,Y): NEXT Y
80 GOTO 80

```

```

10 REM CH. 9, EX. 12, PROBLEM 6
12 CLS
15 X=0: FOR Y=0 TO 42
16 SET(X,Y): NEXT Y
20 X=1: FOR Y=0 TO 42 STEP 6
21 SET(X,Y): NEXT Y
25 Y=42: FOR X=0 TO 127
26 SET(X,Y): NEXT X
30 FOR C=0 TO 1100
35 I=C/2.54
40 X=C: Y=42-I
42 IF Y>0 SET (X,Y)
45 NEXT C
50 GOTO 50

```

```

10 REM CH. 9, EX. 12, PROBLEM 8
11 ON ERROR GOTO 80
12 INPUT "TOTAL NO. OF STEPS"; N
13 X=64: Y=20: K=0: DIST=0
14 CLS: RANDOM
15 SET (X,Y)
16 K=K+1: IF K>N THEN 90
17 Z=RND(2)
18 IF Z=2 THEN 31
21 X=X-2: DIST=DIST-2: GOTO 15
31 X=X+2: DIST=DIST+2: GOTO 15
41 Y=Y+2: GOTO 15
51 X=X-2: GOTO 15
80 N=K
85 GOTO 85
90 PRINT @ 0, "AFTER"; N; "STEPS ";
93 PRINT "DISTANCE ="; DIST; "FEET"
95 GOTO 95

```

```

10 REM CH. 9, EX. 12, PROBLEM 10
12 CLS: REM LABEL Y-AXIS
15 PRINT @ 3, "30": PRINT @ 131, "24"

```

```

16 PRINT @ 259, "18": PRINT @ 387, "12"
17 PRINT @ 515, "6"
19 REM LABEL X-AXIS
20 PRINT @ 704, "2";
22 PRINT "46";
23 PRINT "810"
24 REM DRAW Y-AXIS
25 X=0: FOR Y=0 TO 30
26 SET (X,Y): NEXT Y
27 REM MARKERS ON Y-AXIS
28 X=1: FOR Y=0 TO 30 STEP 3
29 SET (X,Y): NEXT Y
30 REM DRAW X-AXIS
32 Y=30: FOR X=0 TO 100
34 SET (X,Y): NEXT X
35 REM MARKERS ON X-AXIS
36 Y=29: FOR X=0 TO 100 STEP 10
37 SET (X,Y): NEXT X
40 FOR X=0 TO 10
45 Y=X: Y=30-Y
47 XSCALE=10*X
50 SET (XSCALE,Y): NEXT X
55 FOR X=0 TO 10
60 Y=3*X: Y=30-Y
63 XSCALE=10*X
65 SET (XSCALE,Y): NEXT X
70 GOTO 70

```

\*\*\*\*\*

# CHAPTER 10 EXERCISES 13

```

10 REM CH. 10, EX. 13, PROBLEM 6-A
12 CLEAR 500: Z=0
20 INPUT "YOUR TEXT"; T$
30 FOR K=1 TO LEN(T$)
40 IF MID$(T$, K, 1)="E" THEN Z=Z+1
50 E=E+1
60 NEXT K
70 PRINT "THE LETTER E OCCURS"; Z;
73 PRINT "TIMES IN YOUR TEXT"

```

```

10 REM CH. 10, EX. 13, PROBLEM 6-B
12 CLEAR 500: Z=0
20 INPUT "YOUR TEXT"; T$
30 FOR K=1 TO LEN(T$)
40 X$=MID$(T$, K, 1)
42 FOR R=1 TO 5
44 IF X$=MID$("AEIOU", R, 1) THEN Z=Z+1
50 NEXT R
60 NEXT K
70 PRINT "THE VOWELS OCCUR"; Z;
73 PRINT "TIMES IN YOUR TEXT"

```

```

10 REM CH. 10, EX. 13, PROBLEM 6-C
12 CLEAR 500: Z=0
20 INPUT "YOUR TEXT"; T$
30 FOR K=1 TO LEN(T$)
40 IF MID$(T$, K, 4)="ING " THEN Z=Z+1
42 IF MID$(T$, K, 4)="ING." THEN Z=Z+1
60 NEXT K
65 IF RIGHT$(T$, 3)="ING" LET Z=Z+1
70 PRINT "THE ENDING 'ING' OCCURS";
73 PRINT Z; "TIMES IN YOUR TEXT."

```



```

10 REM CH. 10, EX. 13, PROBLEM 8
12 CLEAR 1000
20 INPUT "YOUR TEXT"; A$
30 FOR K=1 TO LEN(A$)
40 X$=MID$(A$, K, 1)
50 IF ASC(X$)>64 AND ASC(X$)<91 THEN 55
52 GOTO 60
55 P$=P$+X$
56 PRINT P$
60 NEXT K
65 REM D$ CONTAINS ONLY LETTERS
70 FOR K=LEN(P$) TO 1 STEP -1
75 B$=B$+MID$(P$, K, 1): NEXT K
80 IF P$=B$ THEN 90
85 PRINT "NOT AN IMPERFECT PALINDROME"
87 END
90 PRINT "IMPERFECT PALINDROME"
92 END

```

```

10 REM CH. 10, EX. 13, PROBLEM 10
20 INPUT "YOUR SENTENCE (NO PERIOD)"; A$
30 FOR K=1 TO LEN(A$)
40 IF MID$(A$, K, 1)="" THEN 60
50 B$=B$+MID$(A$, K, 1)
55 GOTO 80
60 IF LEN(MAX$)<LEN(B$) THEN MAX$=B$
70 B$=""
80 NEXT K
85 IF LEN(MAX$)<LEN(B$) LET MA$=B$
90 PRINT "THE LONGEST WORD IS: "; MA$

```

```

10 REM CH. 10, EX. 13, PROBLEM 12
12 FOR K=1 TO 100
15 A$=STR$(K)
20 FOR L=1 TO LEN(A$)
25 IF "7"=MID$(A$, L, 1) THEN 40
30 NEXT L
35 GOTO 60
40 PRINT K; "BUZZ ";
45 IF INT(K/7)*7<>K THEN 60
50 PRINT "BUZZ ";
60 NEXT K

```

```

10 REM CH. 10, EX. 13, PROBLEM 14
12 PRINT "THE SPECIAL NUMBERS ARE:"
15 FOR K=10 TO 99
20 N1=INT(K/10)
25 N2=K-10*N1
30 IF K<>N1*N1+N2*N2 THEN 45
35 PRINT K;
40 FLAG=1
45 NEXT K
50 IF FLAG=1 END
55 PRINT "THERE ARE NO SUCH NUMBERS"

```

# index

## A

Abbreviation, 162  
    PRINT, 6  
    REM, 29  
ABS function, 116, 117  
Absolute value, 64  
Algorithm, 101  
Alphanumeric character, 162  
AND, 47  
Apostrophe, 29  
Argument, 100, 112  
    dummy, 102  
Arithmetic:  
    functions, 13  
    hierarchy of operations, 14, 49  
    integer, 18  
    trick, 32  
Arithmetic functions:  
    addition, 13  
    division, 13  
    exponentiation, 13  
    multiplication, 13  
    subtraction, 13  
Array, 27, 79  
    largest element, 80  
Ascending order, 53  
ASC function, 143  
ASCII code, 143, 144  
Assignment, 9  
ATN function, 117  
AUTO, 26  
Averaging, 73  
Axes X and Y, 101

## B

Back-up tape, 109  
Bar graph, 116, 135, 141

Binary numbers, 150  
Binomial expansion, 102  
Blanks, 10  
Blinking:  
    block, 155  
    display, 124, 129  
Branching, 46  
BREAK, 26, 40, 67, 68, 89, 145

## C

Cassette input-output, 40, 107  
CBDL function, 117  
Celsius degrees, 21, 40, 133  
Chaining, 56  
Character:  
    alphanumeric, 162  
    blank, 8  
    information, 6, 8  
    strings, 7, 42, 142  
Checkbook balancing, 105  
Checking account, 64, 91  
Christmas Club, 78  
CHR\$ function, 143  
CINT function, 117  
CLEAR, 24  
CLEAR key, 4  
CLEAR *n*, 61, 148  
CLOAD, 41  
CLOAD?, 41  
CLS, 33  
Coding a message, 148  
Comma, 98  
    hanging, 98  
Command mode, 23, 25  
Comments, 28  
Compound interest, 14, 32, 51, 129  
Computer-assisted instruction, 125

Conditional transfer, 54  
     to subroutines, 125  
 CONT, 40, 68, 89  
 Convergence criterion, 130  
 COS function, 117  
 Counter, 66  
     increment, 67  
     test, 67  
 CSAVE, 40  
 CSNG function, 117  
 Cursor, 4, 5, 26

## D

DATA, 93  
 Data validity check, 37, 59, 60, 64  
 Debugging, 37, 85  
 Decisions, 46  
     three-way, 53  
     two-way, 51  
 Declaring variable types, 41  
 DEFDBL, 42  
 Definition statements, 42  
 DEFINT, 41, 42  
 DEFSNG, 42  
 DEFSTR, 42  
 Degree:  
     Celsius, 21, 133  
     days, 44  
     Fahrenheit, 21, 133  
 DELETE, 26  
 DELETE characters, 34  
 DELETE line, 33  
 Dice, 119, 120, 129, 135  
 Digital clock, 129  
 DIM, 79  
 Display:  
     slowdown, 145  
     status message, 145  
 Distance formula, 22  
 Divisibility check, 18  
 Division:  
     sign, 13  
     by zero error, 37, 60  
 Dollar bill change, 61  
 Double-precision, 16, 17, 42  
 Drill and practice, 97  
 Dummy:  
     argument, 102  
     loop, 145  
     record, 109

## E

EDIT, 34  
 Editing, 24, 33  
     examples, 36  
     procedure, 35  
 Edit mode, 34  
 Element, 79  
 Employee:  
     earnings, 111  
     file, 111  
 END, 25, 128  
 ENTER key, 5, 7  
 Equal, 46  
 Equal sign, 11  
 Erase:  
     character, 4  
     line, 4  
     program, 27  
     screen, 4  
 ERL function, 60  
 ERR function, 164  
 ERROR code, 164  
 Errors:  
     10, 37, 60  
     FC, 58  
     language, 37  
     logical, 37  
     messages, 159  
     OD, 94, 145  
     OV, 17  
     SN, 5, 8, 37, 60  
     TM, 149  
 Etch-a-sketch, 154  
 Execution:  
     BREAK, 67, 68, 145  
     halt, 68  
     mode, 24, 25  
     sequence, 46  
 EXP function, 117, 130  
 Exponent, 7  
 Exponent sign:  
     on printer, 13, 14  
     on screen, 13, 14  
 EXTRA IGNORED, 165

## F

Factorial, 78, 120, 130  
 Fahrenheit degrees, 21, 40, 133  
 False statement, 47

FC error, 58  
 Fibonacci sequence, 102  
 File:  
   employee, 111  
   inventory, 110  
 FIX function, 117  
 Flag, 74, 109  
 Flowchart, 49  
   symbols, 50  
 Football, 92  
 Formatting output, 97, 104  
 FOR-NEXT, 70  
 FOR-NEXT loop, 70  
 FRE function, 149  
 Frequency count, 119, 135

## G

Games:  
   buzz, 157  
   etch-a-sketch, 154  
   hi-low, 77, 120  
   shoot the duck, 153  
   video, 153  
 GIGO, 59  
 GOSUB, 122  
 GO TO, 53  
 Graph:  
   bar, 116, 135, 141  
   equation, 110  
   histogram, 116  
   lines, 132, 141  
   metric, 140  
   parabola, 100  
   random, 116, 137, 140  
   rectangle, 140  
   temperature, 133  
 Graphic block, 132  
 Graphic display, 115, 133, 136, 139  
 Graphics, 131, 137  
 Greater than, 46

## H

Halt execution, 68  
 Hanging comma, 98  
 Hanging semicolon, 82  
 Heads and tails, 115  
 Hero's formula, 21  
 Hierarchy of operations, 14, 49

Hi-Lo game, 77, 120  
 Histogram, 116  
 Hypotenuse, 27

## I

IF-GO TO, 63  
 IF-THEN, 54, 66  
 IF-THEN-ELSE, 54, 55  
 IF-THEN GO TO, 63  
 Immediate mode, 24, 25  
 Increment counter, 66, 67  
 Infinite loop, 68, 104, 145  
 Infinite series, 71, 78, 130  
 Information:  
   character, 6, 8  
   numerical, 6  
 Initialize counter, 66, 67, 68  
 INKEY\$ function, 151  
 Inner loop, 82  
 INP function, 165  
 INPUT, 31, 33, 93  
   cassette, 107  
   #-1, 108  
 Integer:  
   arithmetic, 18  
   variable, 16, 17, 41  
 Interest, 23, 27, 38  
   compound, 14, 32  
 INT function, 112  
 Inventory file, 110  
 Iteration, 78  
 Iterative formula, 78

## K

Keyboard, 3  
   response, 31  
 Kitty corner, 27

## L

Language errors, 37, 38  
 Leading plus sign, 104  
 Leading zeros, 17  
 LEFT\$ function, 146  
 Left to right rule, 14  
 LEN function, 146  
 Less than, 46  
 LET, 10

Level I versus Level II, 6, 9, 37  
 Library functions, 112, 117  
 Line number, 23, 25  
     BREAK IN, 39  
 LIST, 25, 39  
 LLIST, 166  
 LOG function, 117, 118  
 Logical errors, 37, 38  
 Logical expressions, 48  
 Logical operations, 47  
 Logical variables, 48  
 Long division, 112  
 Loops:  
     debugging, 85  
     dummy, 145  
     FOR-NEXT, 70  
     IF-THEN, 66  
     infinite, 68, 104, 145  
     inner, 82  
     nested, 82, 91, 101  
     outer, 82  
     structure, 66, 67  
 LPRINT, 167

## M

Magic squares, 84, 91  
 Mailing list, 110  
 Main program, 122  
 Manhattan Island, 21  
 Matrix, 84, 91  
 Mean, 129  
 MEM function, 27, 149  
 MEMORY SIZE?, 3  
 Merge, 91, 158  
 Messages, 28, 29  
     coding, 148  
     decoded, 157  
     error, 159  
 Metric conversion, 16, 140  
 MID\$ function, 146  
 Minus sign, 6  
 Mode, 91  
 Mortgage payment, 21, 38  
 Multiple statement lines, 56  
 Multiple subscripts, 84  
 Multiplication:  
     sign, 13  
     table, 82

## N

Negative numbers, 6  
 Nested loops, 82, 91, 101  
 Nested subroutines, 123, 130  
 NEW, 27, 76  
 NOT, 47  
 Numbers:  
     negative, 6  
     positive, 6  
     scientific notation, 7  
 Numeric constant, 7  
 Numeric keypad, 3  
 Numeric limits on magnitude, 17

## O

OD error, 94, 145  
 ON ERROR GO TO, 59, 65, 137  
 ON-GOSUB, 125  
 ON-GO TO, 57  
 Operations:  
     hierarchy, 14, 49  
     logical, 47  
     relational, 46  
 OR, 47  
 Outer loop, 82  
 OUT function, 167  
 Output, 93  
     cassette, 108  
     formatting, 97  
     zones, 98  
 Overflow, 13  
     error, 17

## P

Palindrome, 147, 157  
 Parabola, 100  
 Parentheses, 14  
     redundant, 15  
 Pascal's triangle, 101  
 PEEK function, 167  
 Picture tree, 31  
 Playing computer, 86, 113  
 POINT function, 137  
 POKE function, 167  
 Population, 45  
 POS function, 102  
 Positive numbers, 6

PRINT, 6, 7  
     abbreviation, 6  
     expanded, 4  
 PRINT TAB, 100, 102, 117  
 PRINT USING, 100, 103, 104  
 PRINT @, 102  
 PRINT #-1, 108  
 Program:  
     clarity, 28  
     debugging, 37, 84  
     delete, 27  
     editing, 33  
     erase, 27  
     line number, 23  
     listing, 25  
     mode, 24  
     readability, 28  
     saving, 40  
     tracing, 86  
     writing, 23  
 Programming:  
     mode, 24, 25  
     trick, 132  
 Pythagorean theorem, 27

## Q

Quadratic equation, 64  
 Quotation marks, 7  
 Quotes, 8, 33

## R

RANDOM, 114  
 Random graphic display, 115  
 Random integer, 114  
 Random number, 114  
 Random number generator, 119  
 Random walk, 137  
 READ, 93  
 READY, 3, 24, 102  
 REDO, 33  
 Relational operations, 46  
 REMark, 28, 29  
 Reserved words, 10, 11, 82, 161  
 RESET function, 137  
 RESTORE, 95  
 RESUME, 59  
 RETURN, 122  
 Right triangle, 27

RIGHT\$ function, 146  
 RND function, 114  
 Roulette, 120  
 Rounding, 113  
     to nearest penny, 19, 103  
 Roundoff, 17  
 Rule of 72, 68, 77, 118  
 RUN, 24, 54

## S

Sales report, 98  
 Saving programs, 40  
 Scientific notation, 7  
 Semicolon, 25  
     hanging, 82  
 Series:  
     exponential, 130  
     Fibonacci, 102  
     infinite, 71, 78, 130  
     sine, 120  
 SET function, 131  
 SGN function, 117  
 SHIFT key, 4  
 SHIFT @, 67, 145  
 Shoot the duck, 153  
 Significant figures, 16, 17, 105  
 Simulate:  
     birthdates, 120  
     dice, 119, 120, 129, 135  
     heads and tails, 115  
     roulette, 120  
 SIN function, 117  
 Single-precision, 16, 17, 42  
     versus double-precision, 18  
 Slope formula, 22  
 Slowdown display, 145  
 Sorting, 86  
 Space bar, 36  
 SQR function, 117  
 Standard deviation, 129  
 State capitals, 95  
 Statement chaining, 56  
 Status message, 145  
 STEP, 70  
 STOP, 39, 89  
 String, 7, 142  
     concatenation, 14, 142  
     entry routine, 152  
     functions, 143, 148  
     variables, 8, 17

STRING\$ function, 148  
 STR\$ function, 149  
 Subroutines, 122  
     conditional transfer, 125  
     nested, 123, 130  
 Subscript, 79  
 Subscripted variables, 79  
     element, 79  
     multiple, 84  
     rules, 80  
     single, 79  
 Syntax error, 5, 8, 37, 60

## T

TAB function, 100, 102, 117  
 TAN function, 117  
 Temperature conversion, 21, 40, 133  
 Three-way decision, 52  
 TM error, 149  
 Trace, 86, 88  
 Trailing plus sign, 104  
 Trailing zeros, 6, 104  
 Transfer:  
     conditional, 54  
     statements, 46, 53  
     unconditional, 53  
 TROFF, 86, 88  
 TRON, 86, 88  
 True statement, 47  
 Two-way decision, 51

## U

Unconditional transfer, 53  
 Underlining a title, 150  
 Unit pricing, 95  
 USR function, 170

## V

VAL function, 149  
 Variable, 8  
     declare, 17, 41  
     display, 58  
     double-precision, 16, 17, 42  
     integer, 16, 17, 42  
     interchange values, 11  
     MEM, 27  
     numerical, 9  
     single-precision, 16, 17, 42  
     string, 8, 17, 42  
     subscripted, 79, 84  
     types, 16, 41  
 Variable displays, 58  
 VARPTR function, 170  
 Video game, 153

## W

Wise old man, 75

## Y

Year-to-date earnings, 108

## Z

Zero:  
     division by, 37  
     leading, 17  
     subscript, 80  
     trailing, 6, 104  
 Zone, 98



introduction to

---

**TRS-80  
LEVEL II BASIC**

---

and  
computer  
programming

This is an easy-to-follow book suitable for a wide variety of readers who are new to computers and computer programming. It is ideal for the beginner who wants to learn about computers without wishing to become an expert. Dr. Zabinski's INTRODUCTION TO TRS-80 LEVEL II BASIC AND COMPUTER PROGRAMMING shows how to use the TRS-80 computer for a wide range of exciting applications such as checkbook balancing, computer graphics, multiplication tables, magic squares, and video games.

BASIC has a conversational, interactive nature and a simple structure; it is an attractive teaching tool with which computer programming concepts can be presented to beginners. With a small set of instructions, the beginner can very quickly begin to write elementary computer programs—a stimulating and enjoyable experience. Many practical examples are included to illustrate the use of BASIC and to demonstrate how a computer can be programmed.

The opening chapter describes major characteristics of the Radio Shack TRS-80, its keyboard, and how to communicate with the computer. Subsequent chapters discuss, in direct and easy-to-follow language, techniques for specifying information, computer programs, decisions, looping, input and output, library functions, subroutines, graphics, and strings.

Numerous illustrative examples and an abundance of chapter exercises [over 200!], including many of their solutions, are provided. They help the reader assess progress, reinforce comprehension, and provide valuable practical experience.



By Michael P. Zabinski, Ph.D.  
Fairfield University  
Fairfield, Connecticut

Michael P. Zabinski, Ph.D., is a professor at Fairfield University, Fairfield, Connecticut, and a founder and director of the only computer camp for youngsters in the United States. He is a consultant to public schools on computer usage in the classroom. Dr. Zabinski is author of programming books as well as educational materials for Radio Shack.